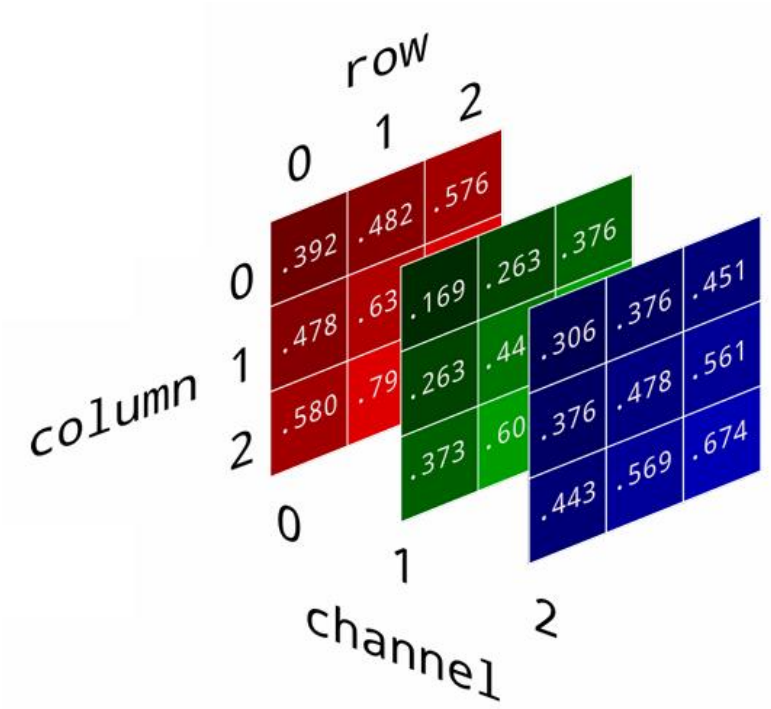
GDSC AI Week

고려대학교 정보대학
Korea University
College of Informatics

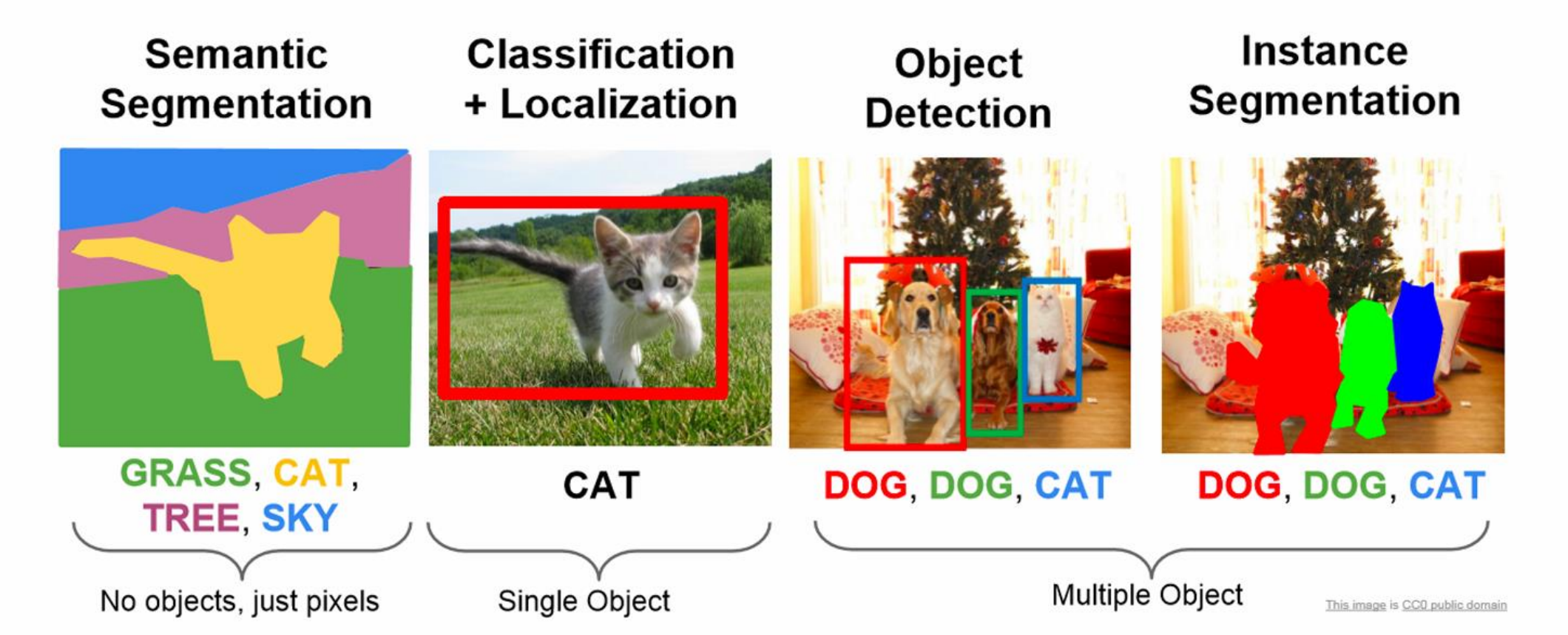
전병우
ipcs@korea.ac.kr

Understanding Image Data

- 우리는 지금까지 1D type의 데이터 포인트만 살펴보았다.
  - 이제 2D-type의 이미지 데이터를 살펴보자.
- Color images는 2D numpy arrays의 집합으로 표현된다: 각 픽셀이 세 개의 값(R, G, B)를 갖는다.
  - Red, Green, and Blue
  - 해상도 1080x720의 이미지 -> 3x1080x720
- 만약 RGB 값을 평균한다면 어떻게 될까?

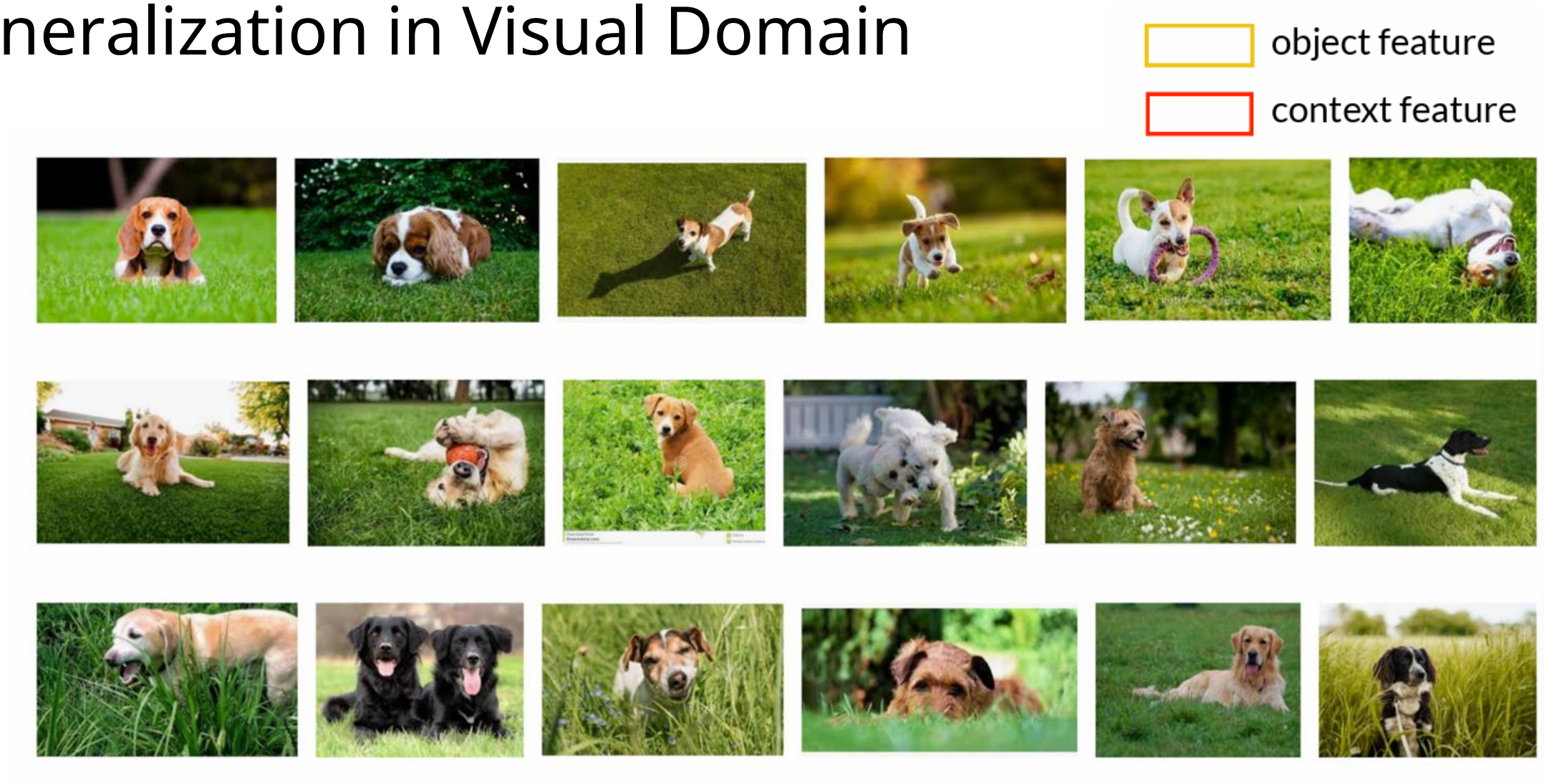


Computer Vision Tasks

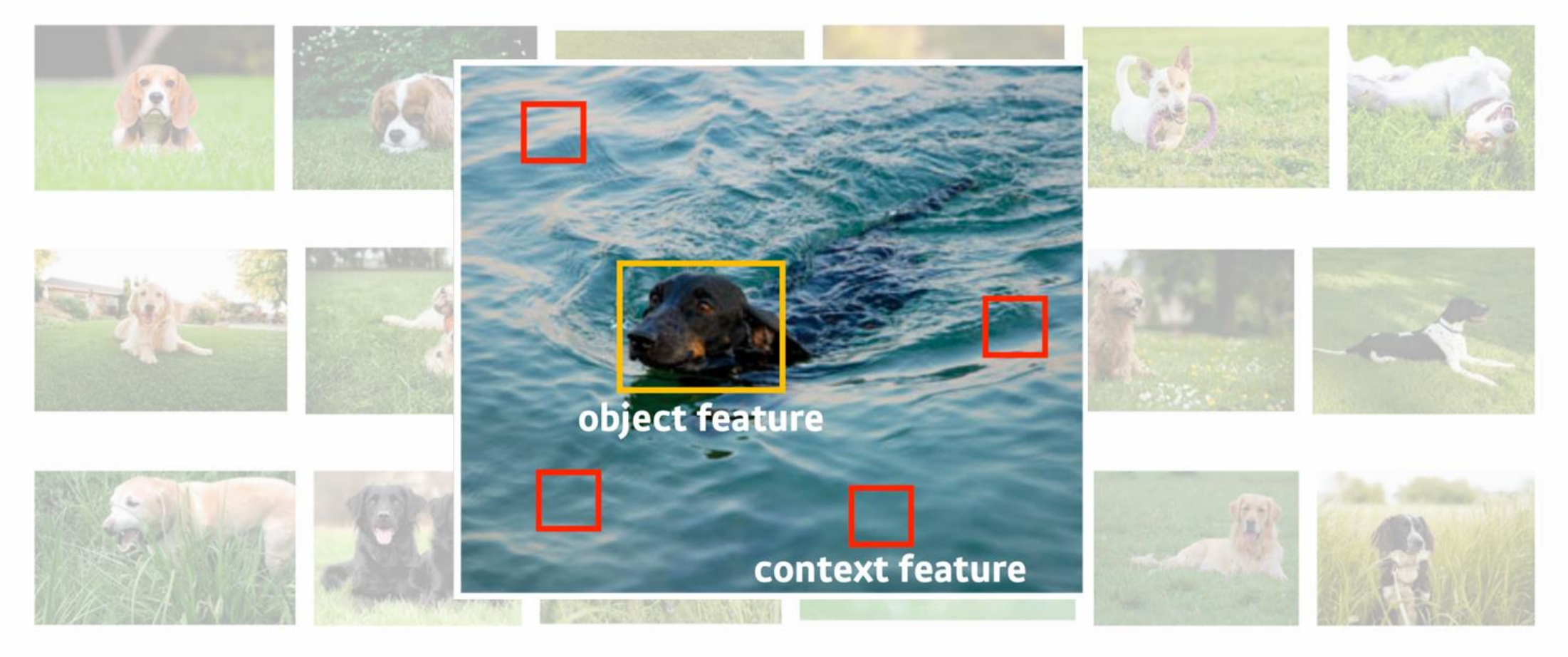


Fei-Fei Li (2017)

Generalization in Visual Domain

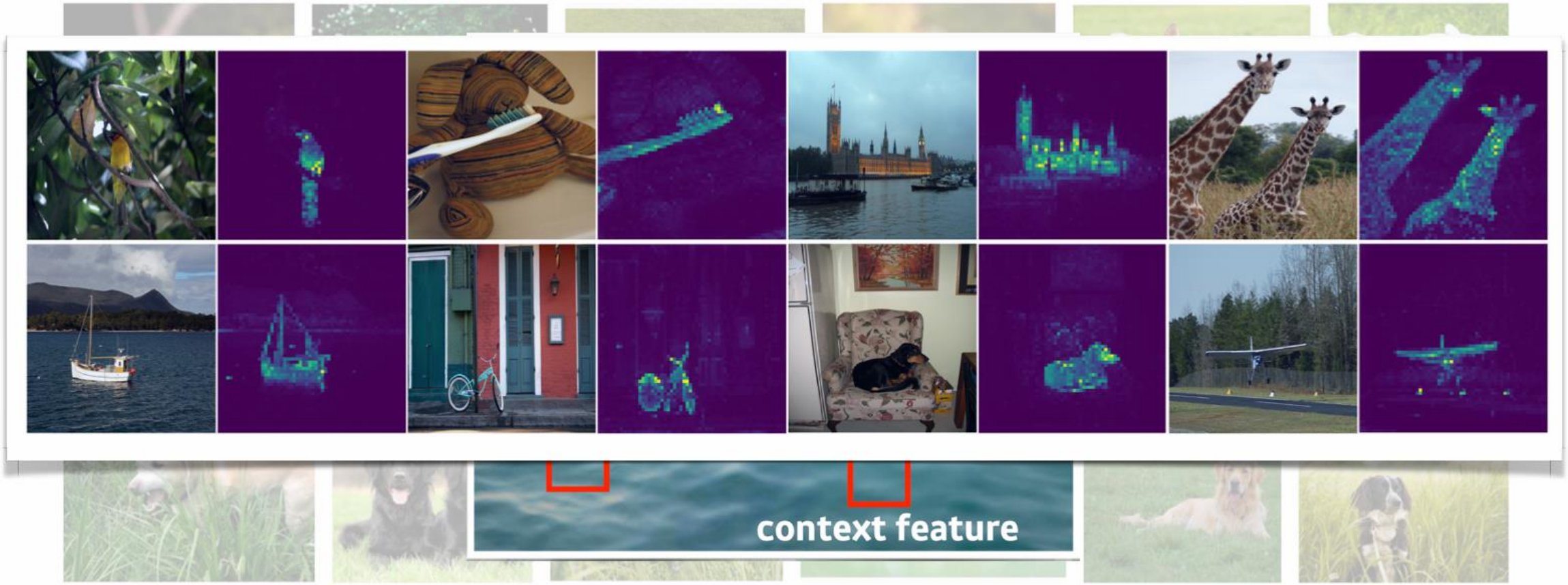


Generalization in Visual Domain



모든 픽셀의 정보를 이용하면 가짜 상관 관계에 취약할 가능성이 더 높다.
ex) 만약 “개”의 정보가 아닌 배경의 정보로만 “개”를 판별했다면?

Generalization in Visual Domain



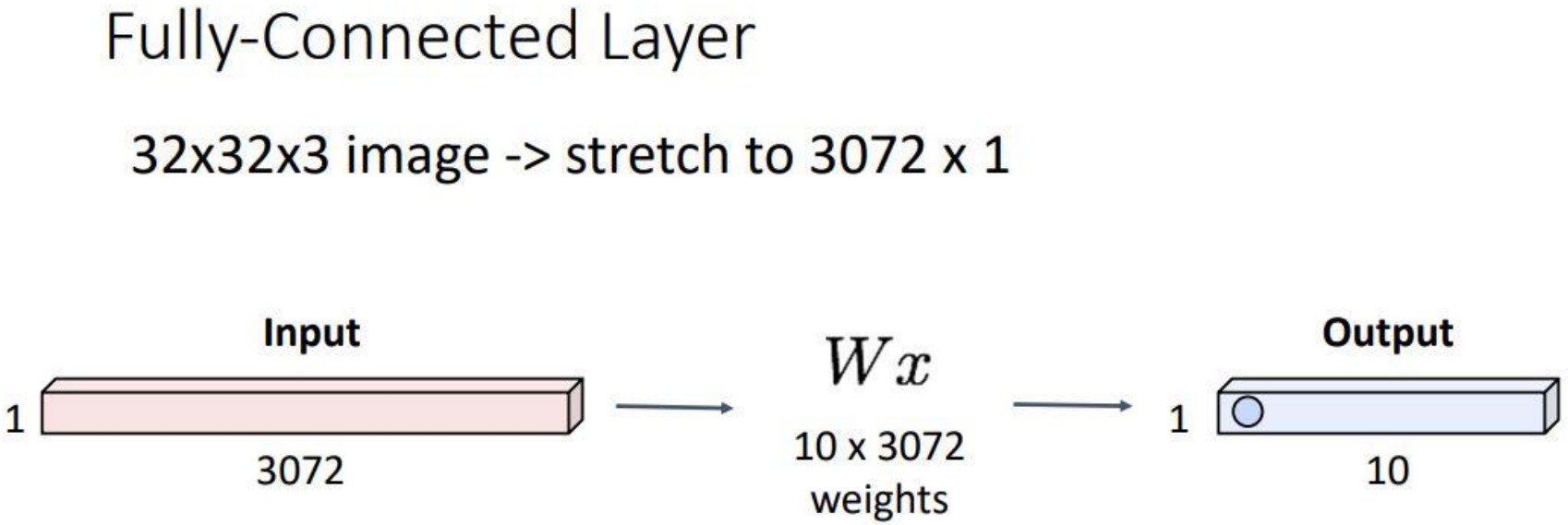
따라서 visual domain의 현대 딥러닝은 object features를 robustly extracting 한다

어떤 모델이 더 interpretable할까?

Principles for Visual Domain

- Locality principle
  - 멀리 떨어져 있는 영역끼리는 irrelevant할 것이다.

Why not MLP?

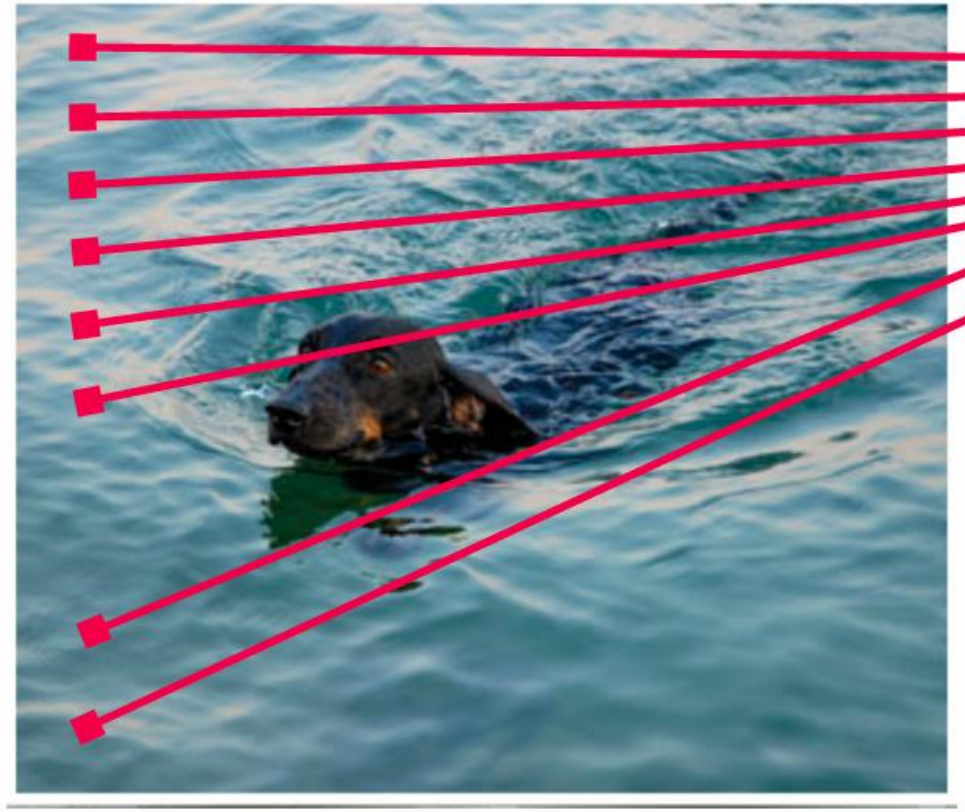


문제점?

1. Locality principle을 반영하지 못한다.
2. 이미지에 있는 객체의 위치가 달라지면?
3. 이미지의 크기가 달라지면?

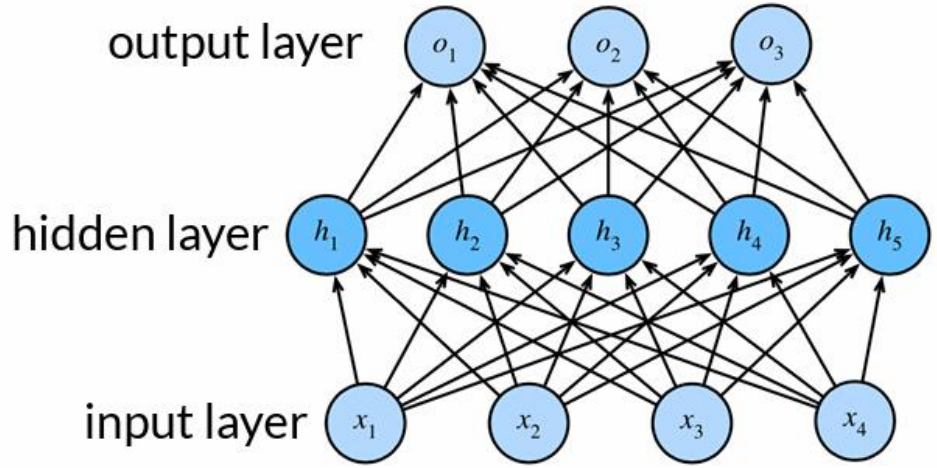
Why not MLP?

- memory inefficient
- hard to optimize
- prone to overfitting



$$h_i = \sum_j w_{ij} x_j$$

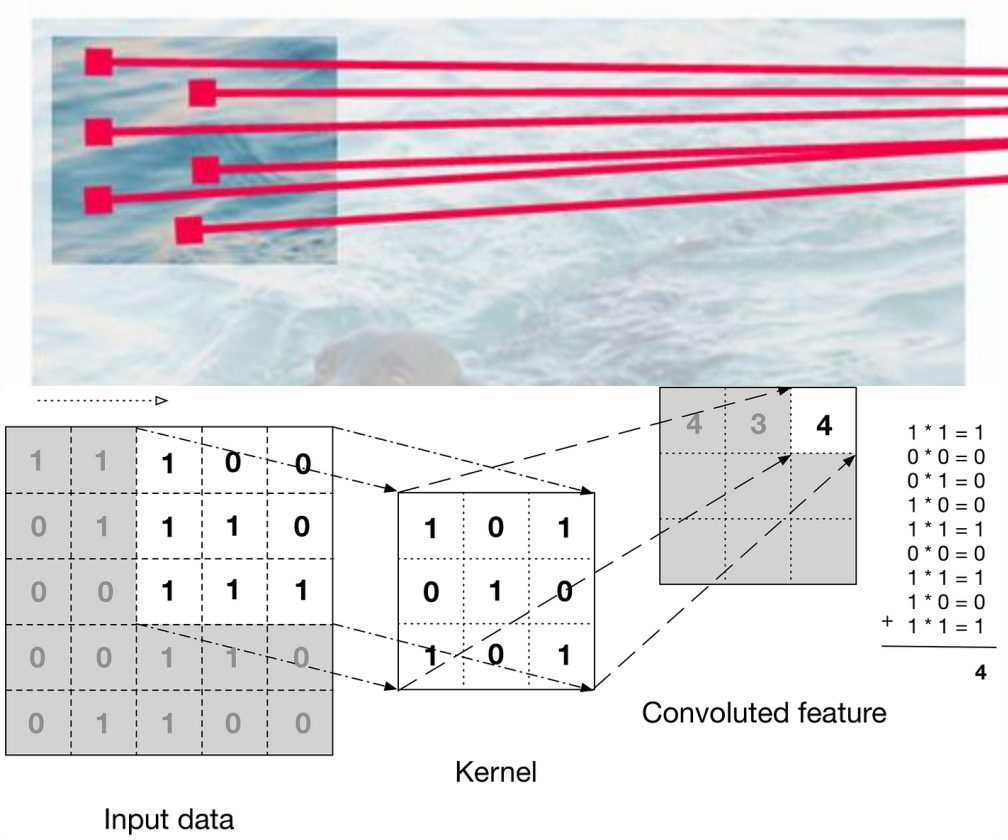
In MLP, every pixel has own parameters



1. Locality Principle

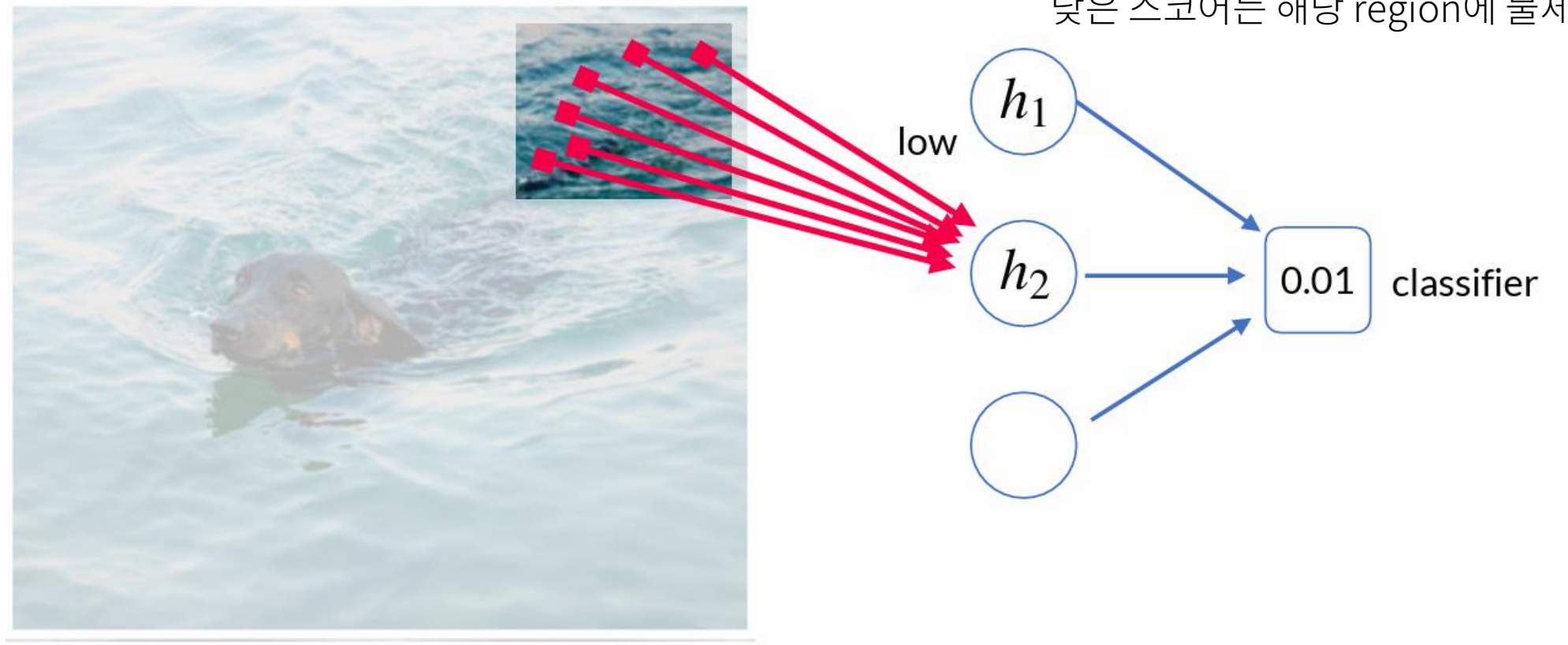
Locality Principle

Idea :
이미지를 패치(patch) 단위로 나누어, 스코어를 부여합니다.

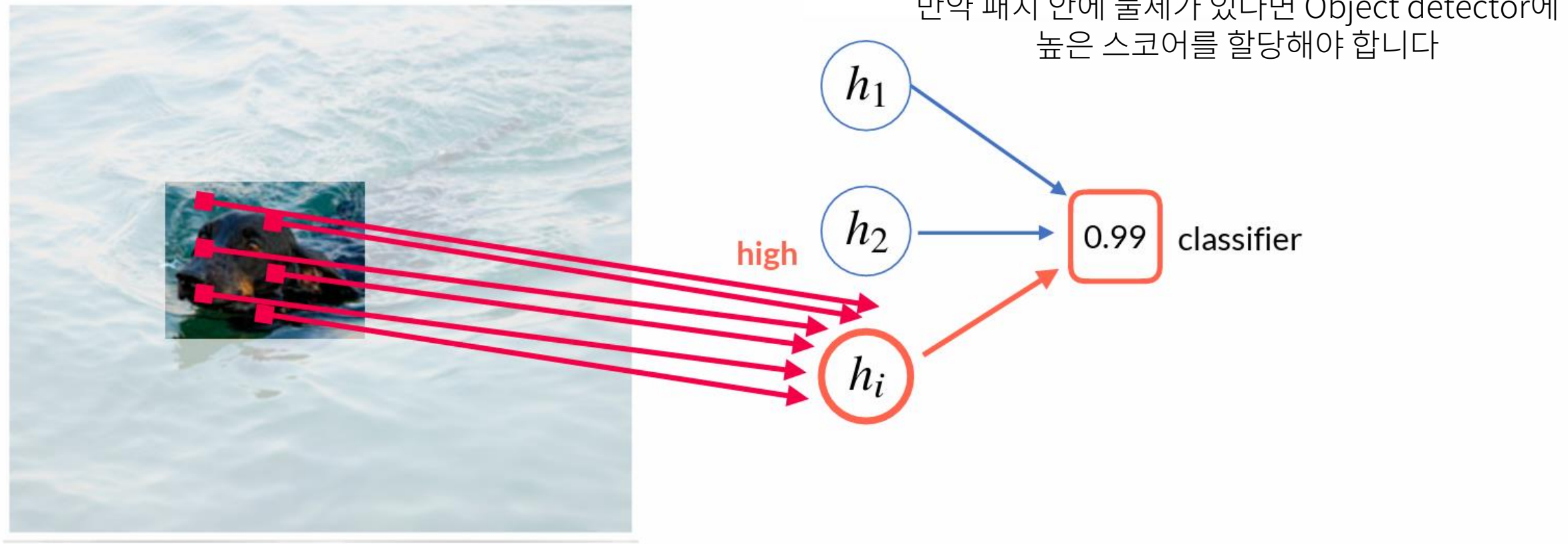


해당 이미지에 개가 있는 지 없는 지 판단하는,
Binary Classifier

Locality Principle



Locality Principle



Principles for Visual Domain

- Locality Principle
  - 멀리 떨어져 있는 영역끼리는 irrelevant할 것이다.
- Spatial Invariance
  - 우리의 visual recognition system은 물체의 위치에 dependent하면 안된다.
  - What object looks like does not depend upon where the object is located
  - translation equivariance
  - translation invariance

translation invariance

$$g_\phi(\mathcal{T}[\mathbf{x}]) = g_\phi(\mathbf{x})$$

translation equivariance

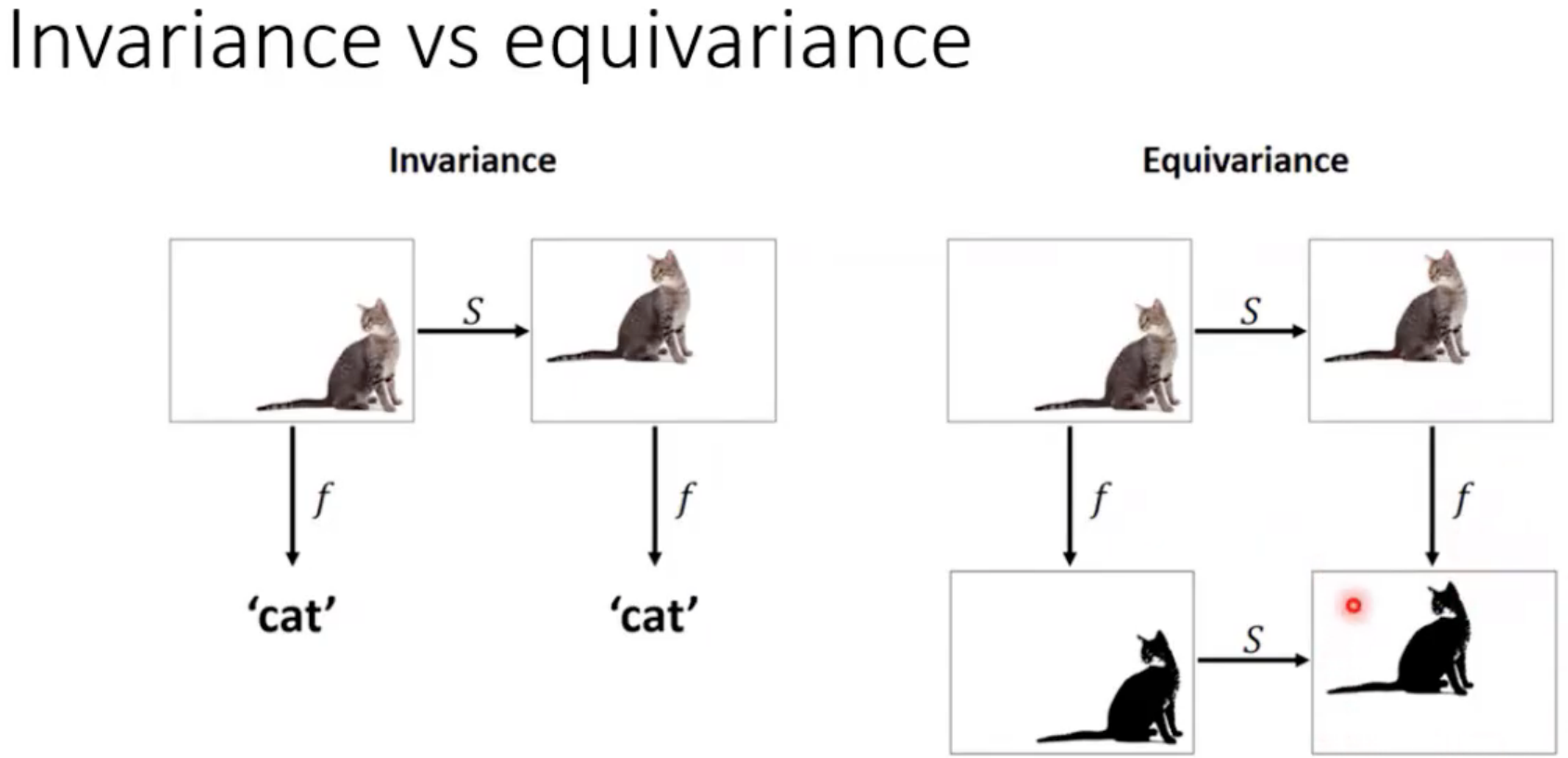
$$f_\theta(\mathcal{T}[\mathbf{x}]) = \mathcal{T}[f_\theta(\mathbf{x})]$$

$\mathcal{T}$ translation operation

$f_\theta,\ g_\phi$ models

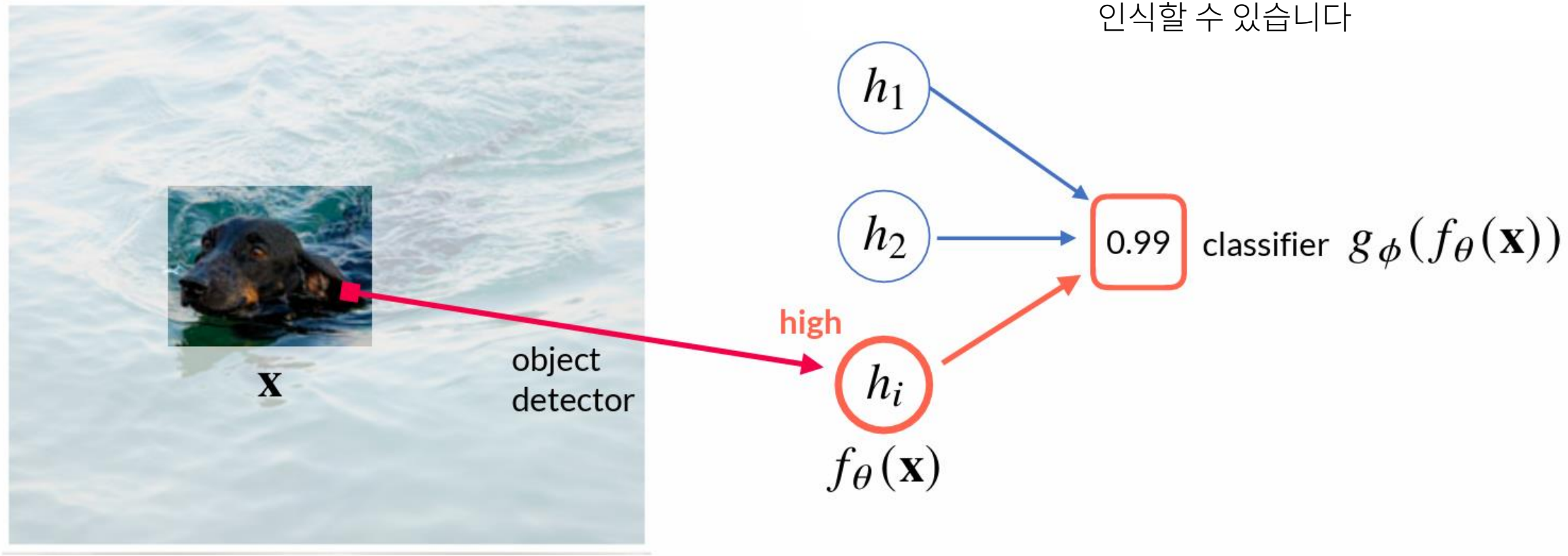
Principles for Visual Domain

- Spatial Invariance



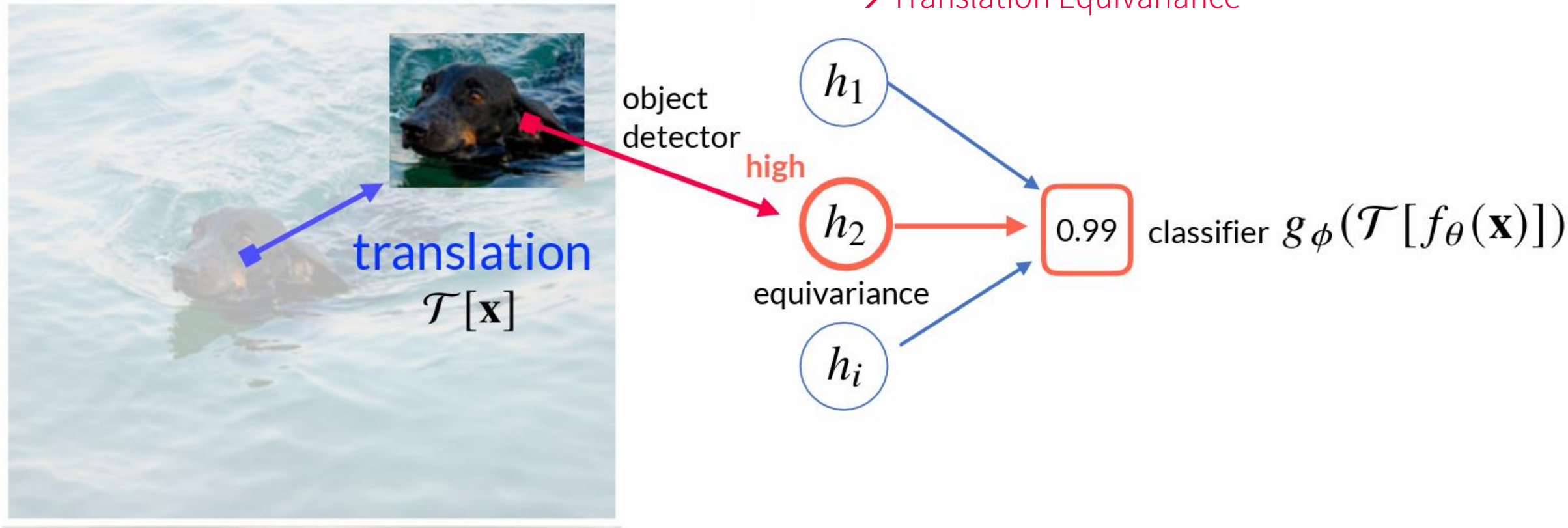
Locality Principle

Locality principle 덕분에 object detector f_θ는 target object가 등장하는 패치의 visual pattern을 인식할 수 있습니다



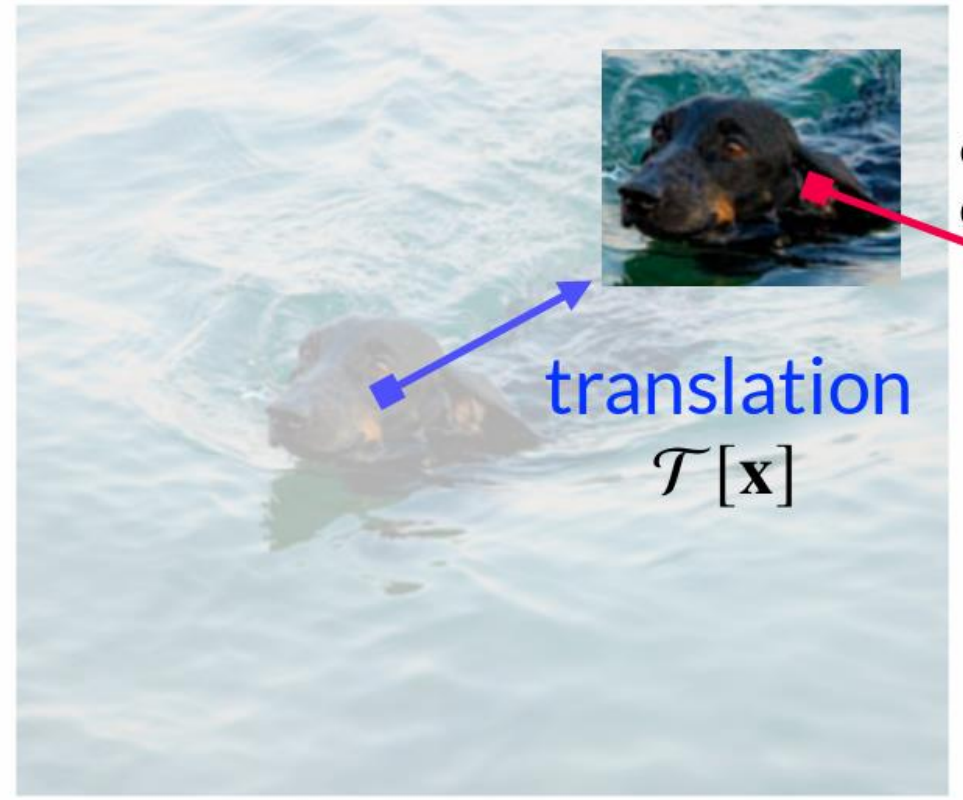
Translation Equivariance

만약 물체의 위치가 이동했다면(translated)
1. f_θ는 이동된 패치(translated patch)에 대해서도 정보를 잡을 수 있어야 합니다 (h_i -> h_2)
→ Translation Equivariance



Translation Equivariance + Invariance

그러면 우리의 classifier g_ϕ의 결과는
이동된 패치에 대해서도 이전과 같아야 합니다
→ Translation Invariance



object detector

high

h_1

h_2

h_i

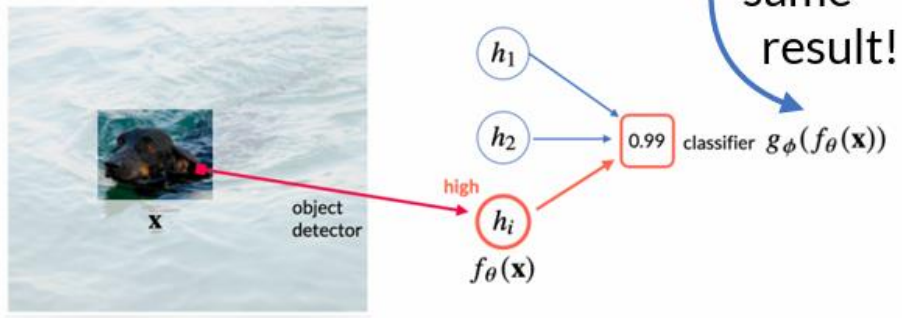
equivariance

invariant aggregation

0.99

classifier $g_\phi(\mathcal{T}[f_\theta(\mathbf{x})])$
$= g_\phi(f_\theta(\mathbf{x}))$

same result!



MLPs are inefficient for spatial invariance

- Fully connected layer

$$h(i) = \sum_{\ell} W_i(\ell) x(\ell)$$

for each current location i, there exists the corresponding weight matrix W_i

location at the current layer

location at the previous layer
(근처, Patch)

MLPs are inefficient for spatial invariance

- Fully connected layer

$$V_i(p) := W_i(i+p)$$

$$h(i) = \sum_{\ell} W_i(\ell)x(\ell) = \sum_{p} V_i(p)x(i+p)$$

weight matrix from the pixel p to the current location i

pixel at the previous layer

location at the current layer

location at the previous layer

we need to train $V_i(\cdot)$ for each current location i

MLPs are inefficient for spatial invariance

- Fully connected layer

$$V_i(p) := W_i(i+p)$$

$$h(i) = \sum_{\ell} W_i(\ell)x(\ell) = \sum_{p} V_i(p)x(i+p)$$

location at the current layer

location at the previous layer

이렇게 정의할 경우,
i에 따라 V_i(p)는 달라짐.

weight matrix from the pixel p to the current location i

pixel at the previous layer

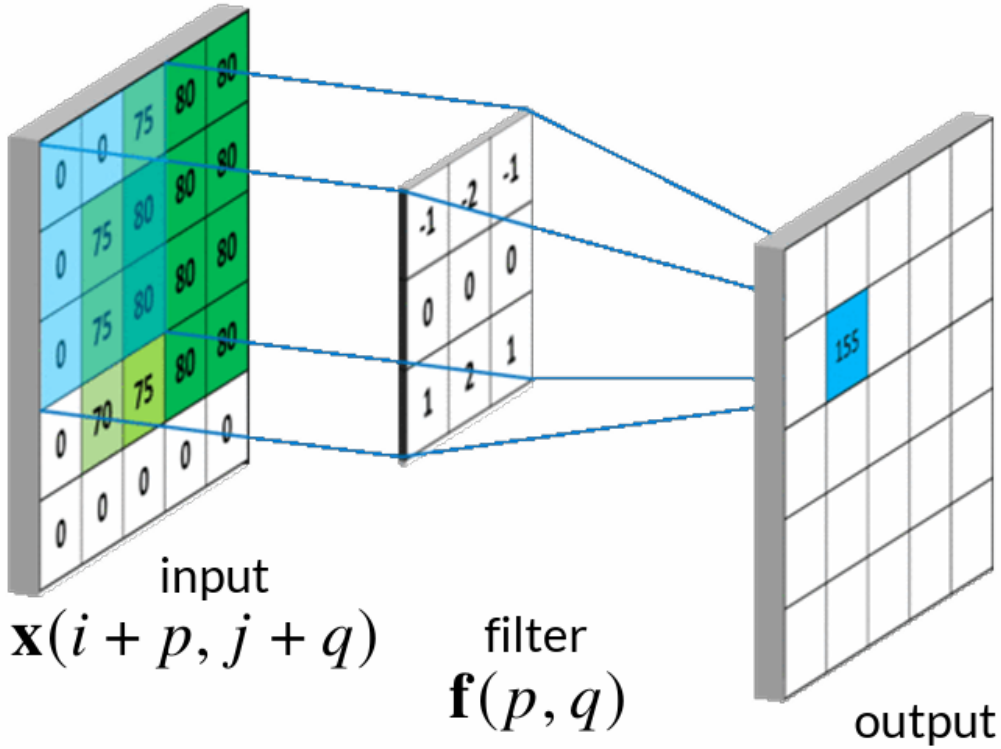
- 여기서 합성곱(convolution) 연산을 사용해보자

convolution shares parameters $V(\cdot)$ along the current locations i
→ spatial invariance

$$h(i) = \sum_{p} V(p)x(i+p)$$

$|p| \le \Delta$ patch size → locality principle

What is Convolution?



1D Conv $$[\mathbf{f} * \mathbf{x}](i) = \sum_{p} \mathbf{f}(p)\mathbf{x}(i+p)$$

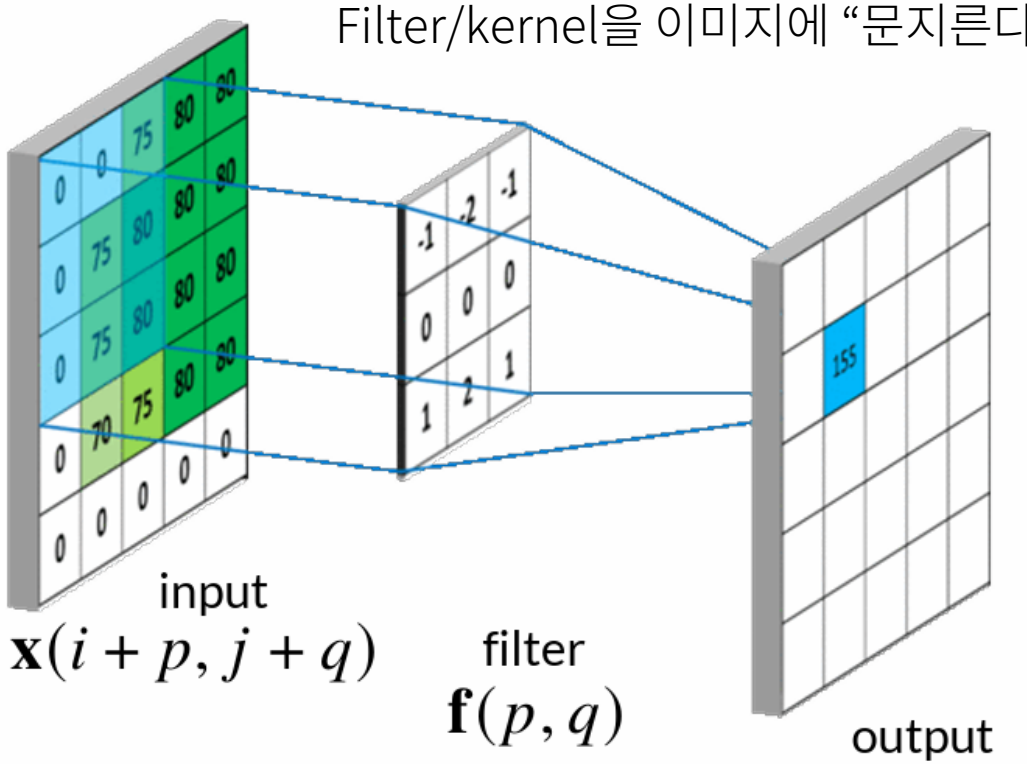
2D Conv $$[\mathbf{f} * \mathbf{x}](i, j) = \sum_{p,q} \mathbf{f}(p, q)\mathbf{x}(i+p, j+q)$$

3D Conv $$[\mathbf{f} * \mathbf{x}](i, j, k) = \sum_{p,q,r} \mathbf{f}(p, q, r)\mathbf{x}(i+p, j+q, k+r)$$

What is Convolution?

Filter/kernel을 이미지에 “문지른다”

필터(filter, kernel)는 입력의 위치와 무관(invariant)하다



1D Conv $$[\mathbf{f} * \mathbf{x}](i) = \sum_{p} \mathbf{f}(p)\mathbf{x}(i+p)$$

2D Conv $$[\mathbf{f} * \mathbf{x}](i,j) = \sum_{p,q} \mathbf{f}(p,q)\mathbf{x}(i+p, j+q)$$

3D Conv $$[\mathbf{f} * \mathbf{x}](i,j,k) = \sum_{p,q,r} \mathbf{f}(p,q,r)\mathbf{x}(i+p, j+q, k+r)$$

What exactly filters do?

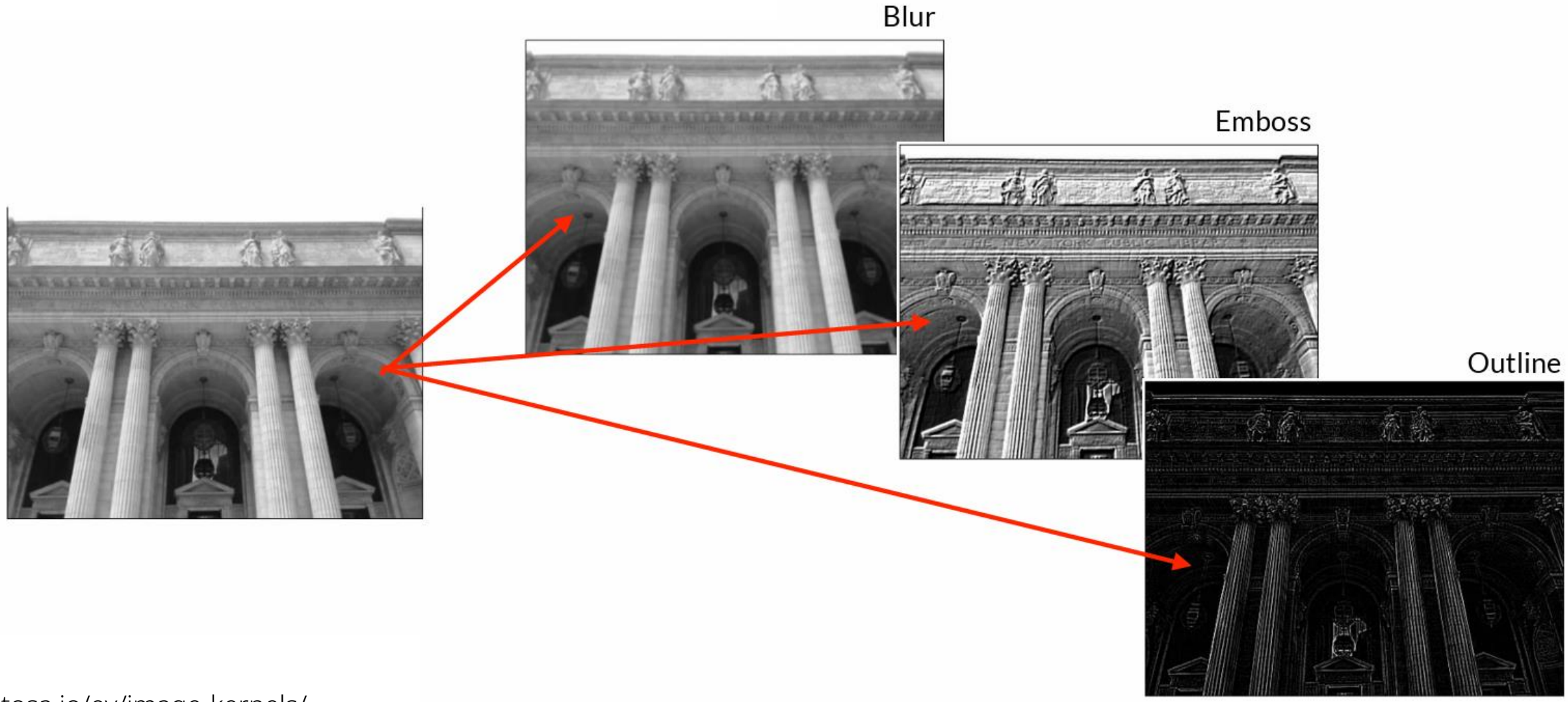
0.06	0.12	0.06
0.12	0.25	0.12
0.06	0.12	0.06

-2	-1	0
-1	1	1
0	1	2

-1	-1	-1
-1	8	-1
-1	-1	-1

https://setosa.io/ev/image-kernels/

What exactly filters do?



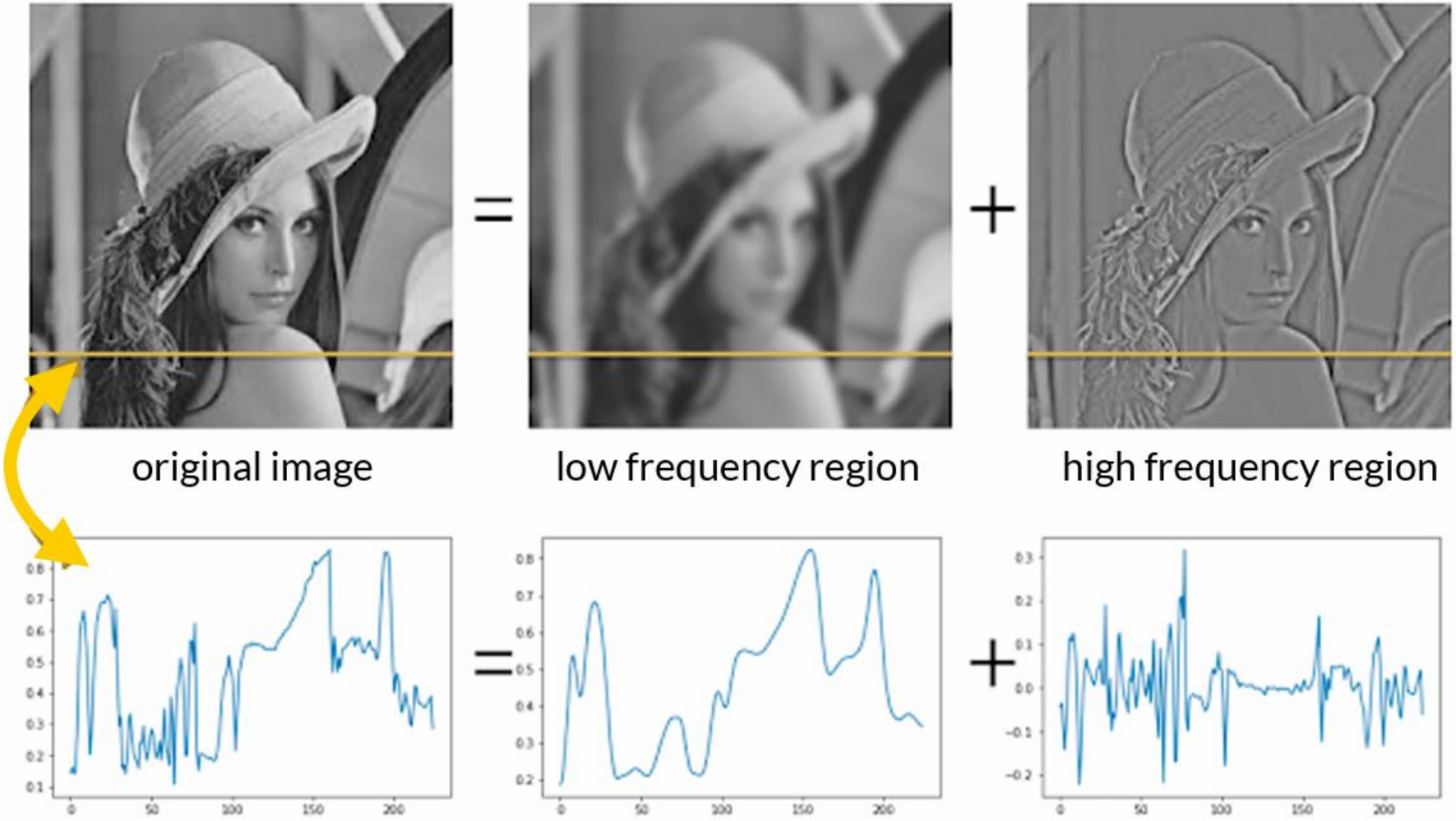
https://setosa.io/ev/image-kernels/

Convolution and Fourier Transform

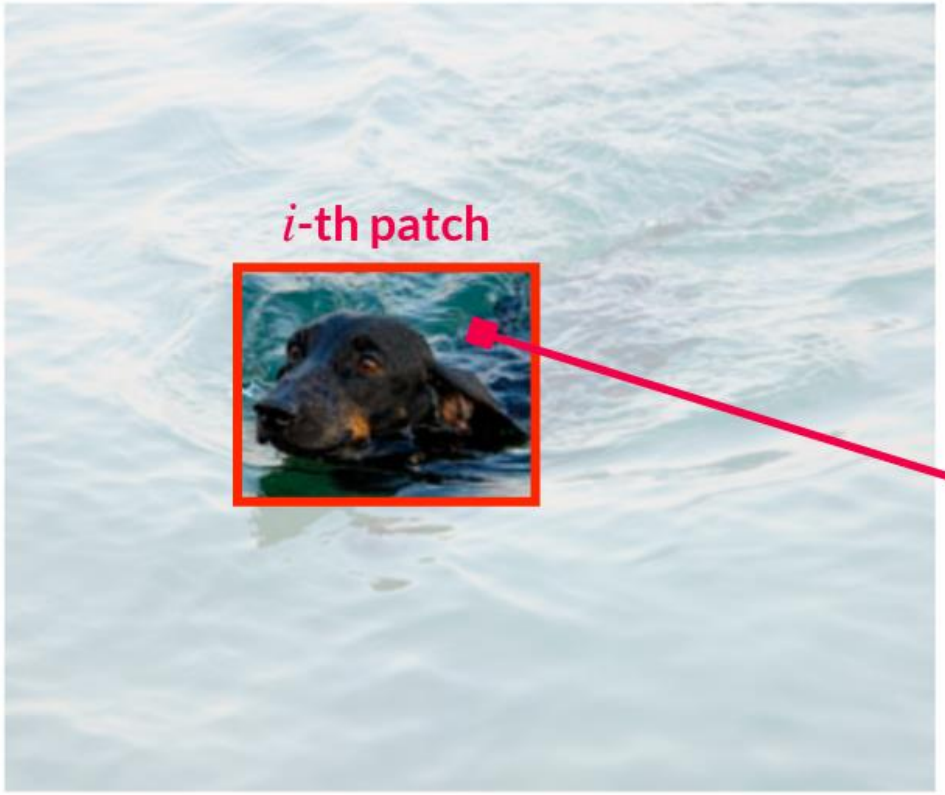
$$\mathcal{F}[\mathbf{f} * \mathbf{x}] = \mathcal{F}[\mathbf{f}]\mathcal{F}[\mathbf{x}]$$

convolution filter image

다양한 필터는 다양한 주파수의
시그널을 포착하고 강화합니다



How convolution operations satisfy principles?



$$h_1 = \sum_{|p|\leq\Delta} \mathbf{f}(p)\mathbf{x}(1+p) \in \text{1st patch}$$

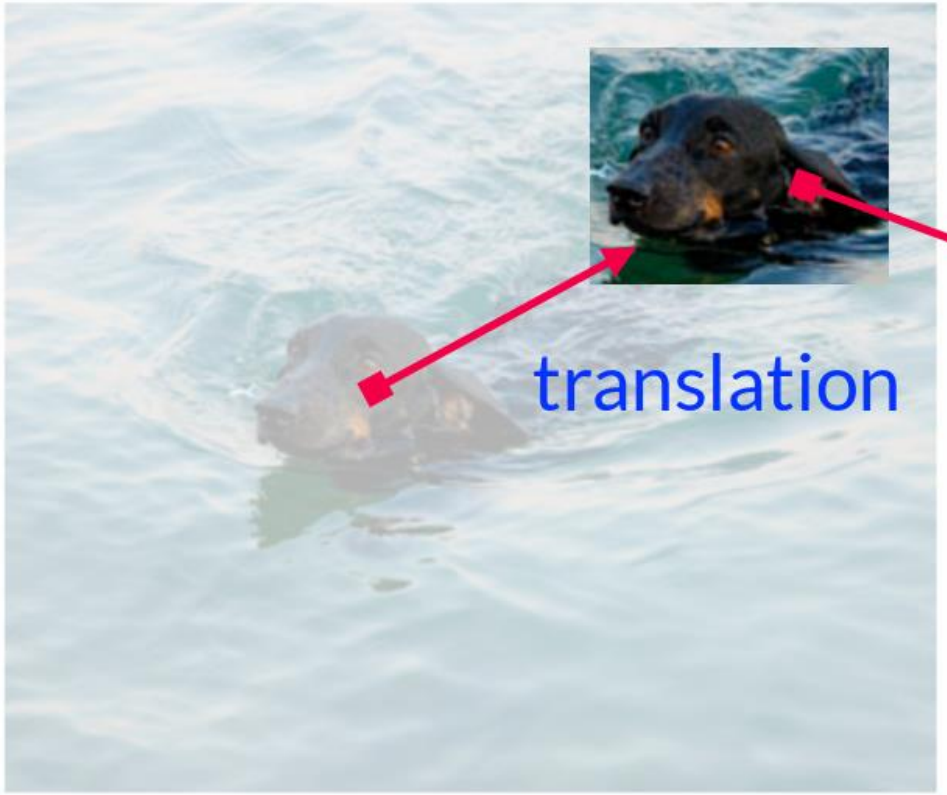
$$h_2 = \sum_{|p|\leq\Delta} \mathbf{f}(p)\mathbf{x}(2+p) \in \text{2nd patch}$$

$$\vdots \qquad \vdots \qquad \vdots$$

$$h_i = \sum_{|p|\leq\Delta} \mathbf{f}(p)\mathbf{x}(i+p) \in i\text{-th patch}$$

Convolution 연산은 local patch를 통해 시각 정보를 처리합니다
=> Locality principle

How convolution operations satisfy principles?



합성곱은 같은 파라미터를 서로 다른 패치에 적용합니다
=> Translation Equivariance

$$h_1 = \sum_{|p|\le\Delta} \mathbf{f}(p)\mathbf{x}(1+p) \in \text{1st patch}$$

$$h_2 = \sum_{|p|\le\Delta} \mathbf{f}(p)\mathbf{x}(2+p) \in \text{2nd patch}$$

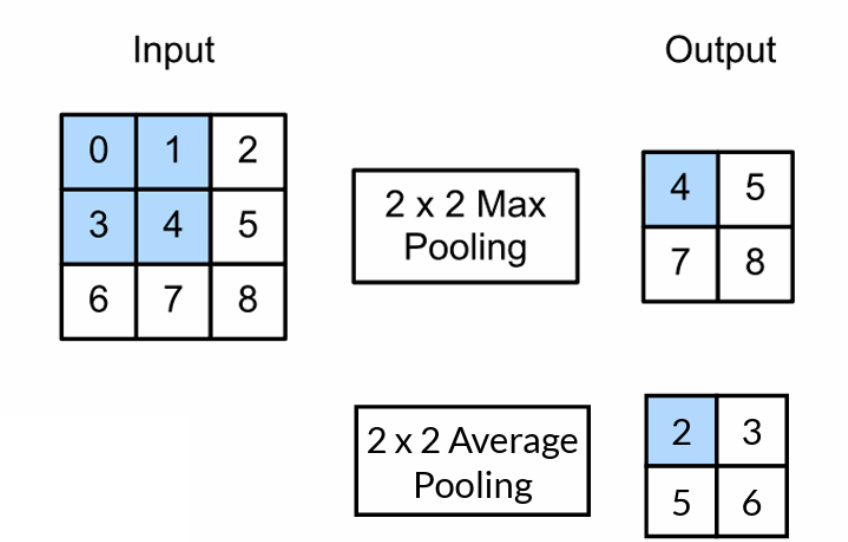
$$\vdots \qquad \vdots \qquad \vdots$$

$$h_i = \sum_{|p|\le\Delta} \mathbf{f}(p)\mathbf{x}(i+p) \in i\text{-th patch}$$

filters

Pooling

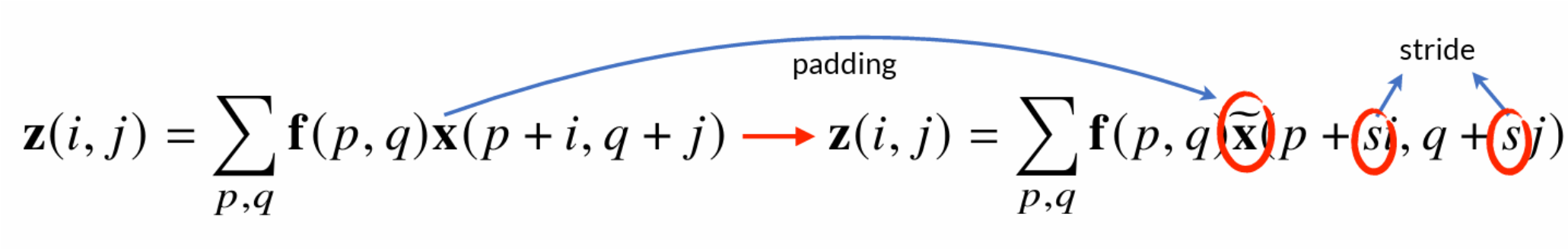
- Pooling을 통해 조금씩 공간 해상도(spatial resolution)를 줄여 나간다.
- Pooling은 translation invariance를 만족하는 연산이다.
  - aggregate information
  - downsampling
  - parameter-free operator
  - locally translation invariance
- 대표적인 Invariant aggregation은 두 가지가 있다.
  - Average pooling
  - Max pooling



(Pop-quiz) Conv + Activation + Pooling일까, Conv + Pooling + Activation일까?

Stride & Padding

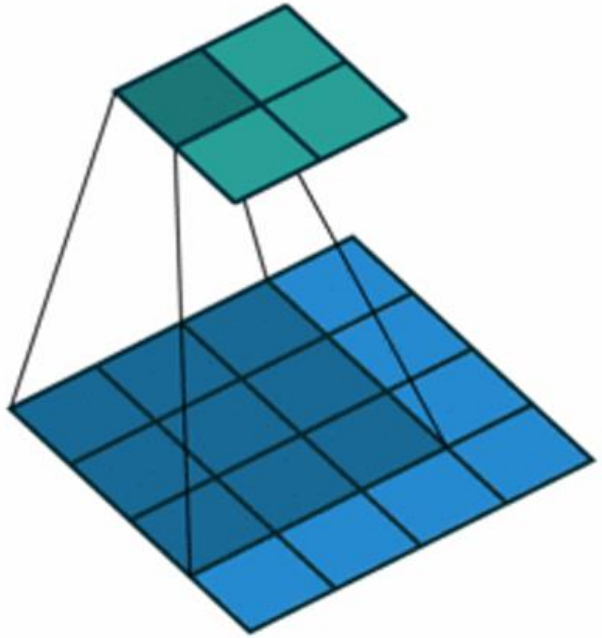
- 필터(커널)은 보통 width와 height가 1보다 크다.
  - 합성곱 연산을 반복하면 입력보다 상당히 작은 출력이 나오는 경향이 있다.
  - Padding은 이를 방지하는 가장 일반적인 방법이다.
    - 만약 출력 차원을 빠르게 줄이고 싶다면 stride를 사용한다.



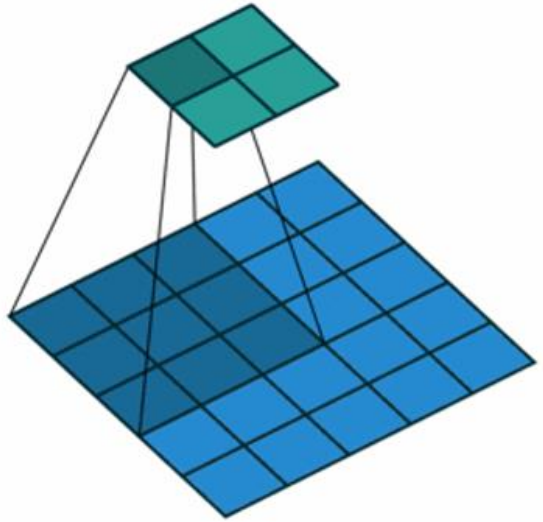
Stride & Padding

(k_h, k_w) filter size (p_h, p_w) padding size (s_h, s_w) stride size

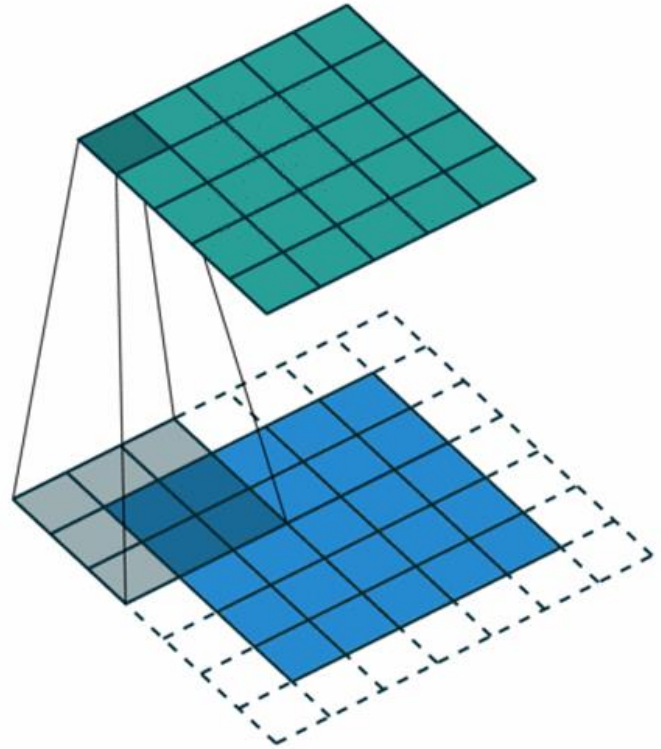
$$(h_{\text{out}}, w_{\text{out}}) = \left(\left\lfloor \frac{h_{\text{in}} - k_h + 2p_h}{s_h} \right\rfloor + 1, \left\lfloor \frac{w_{\text{in}} - k_w + 2p_w}{s_w} \right\rfloor + 1 \right)$$



Vanilla convolution



convolution + stride



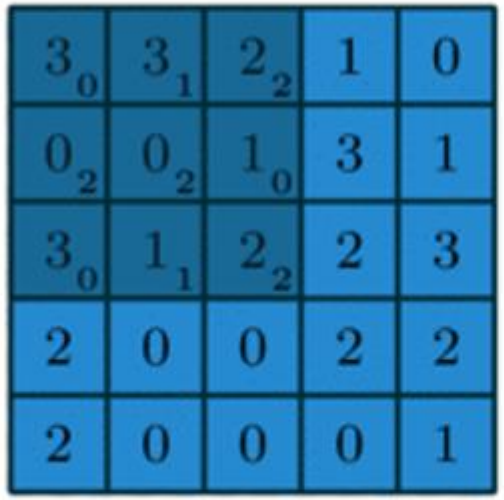
convolution + padding

Stride & Padding

(k_h, k_w) filter size (p_h, p_w) padding size (s_h, s_w) stride size

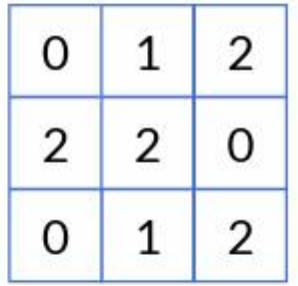
$$(h_{\text{out}}, w_{\text{out}}) = \left(\left\lfloor \frac{h_{\text{in}} - k_h + 2p_h}{s_h} \right\rfloor + 1, \left\lfloor \frac{w_{\text{in}} - k_w + 2p_w}{s_w} \right\rfloor + 1 \right)$$

- 예시 : 3x3 convolution



input * filter

output



filter

3x3

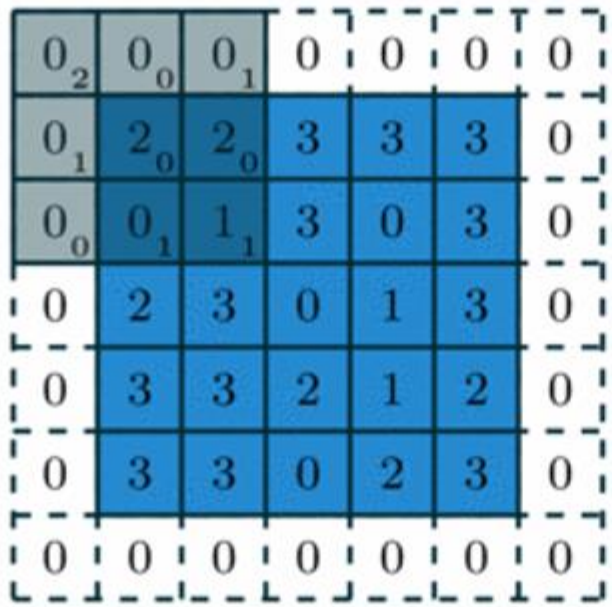
$$\left\lfloor \frac{5 - 3 + 0}{1} \right\rfloor + 1 = 3$$

Stride & Padding

(k_h, k_w) filter size (p_h, p_w) padding size (s_h, s_w) stride size

$$(h_{\text{out}}, w_{\text{out}}) = \left(\left\lfloor \frac{h_{\text{in}} - k_h + 2p_h}{s_h} \right\rfloor + 1, \left\lfloor \frac{w_{\text{in}} - k_w + 2p_w}{s_w} \right\rfloor + 1 \right)$$

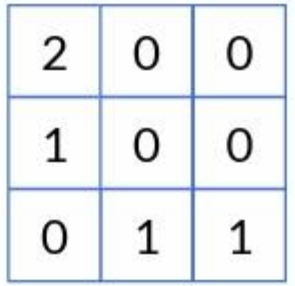
- 예시 : 3x3 convolution + 1x1 zero padding + 2x2 stride



input * filter

output

3x3

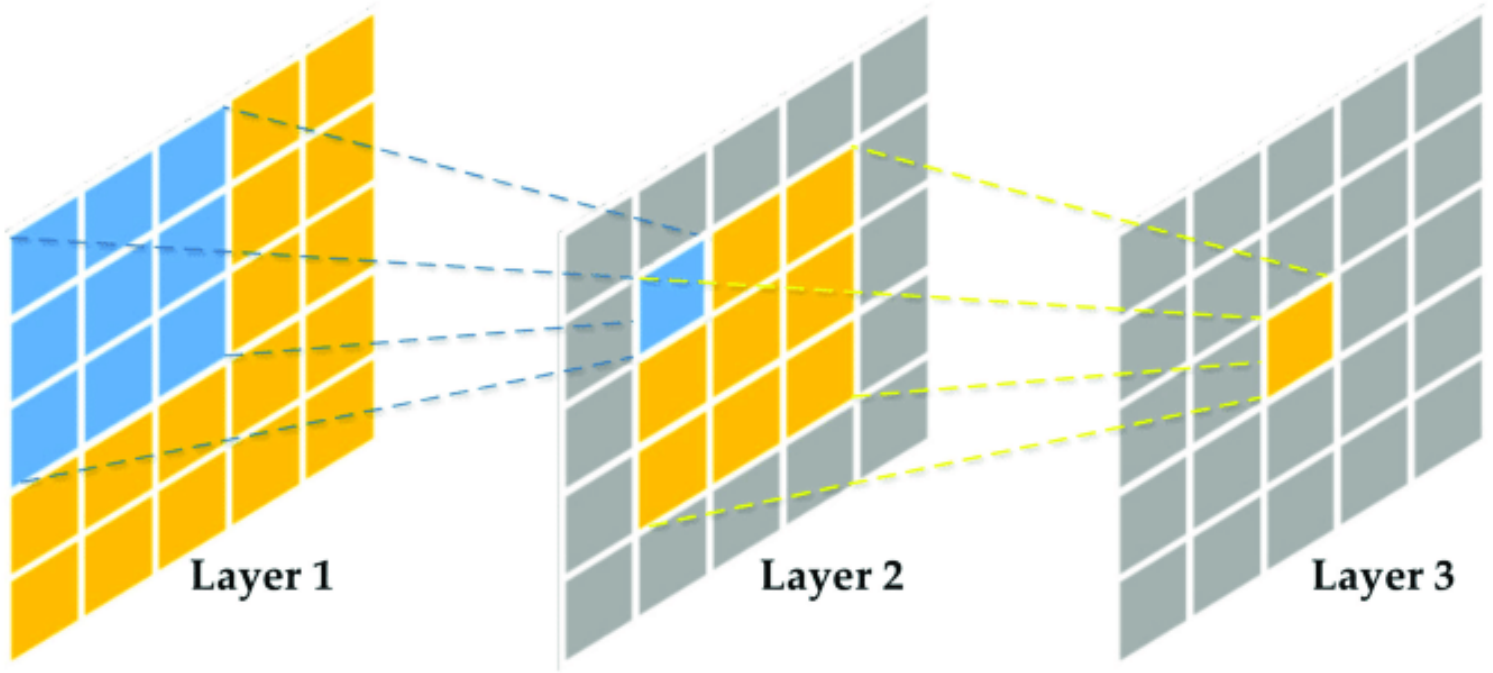


filter

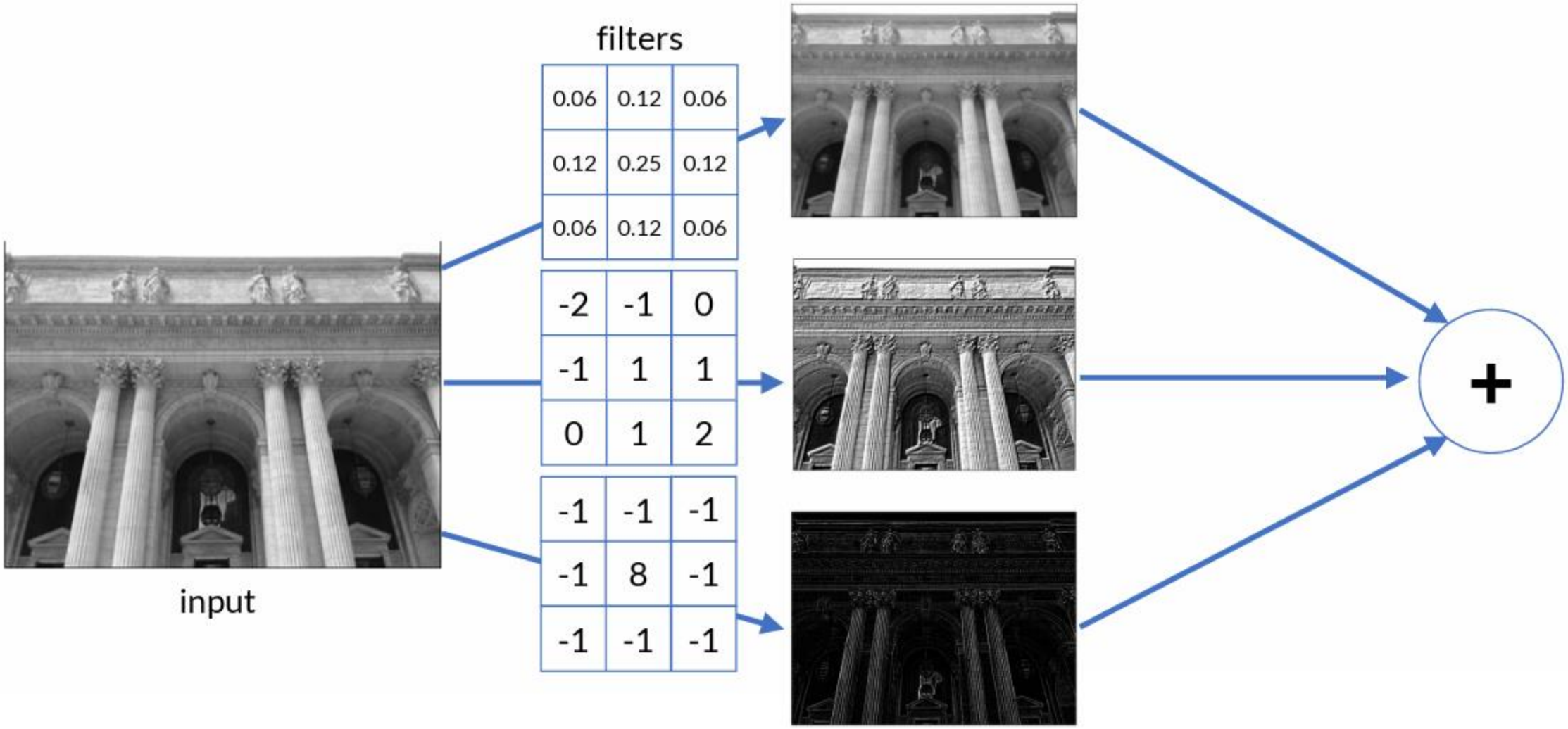
$$\left\lfloor \frac{5 - 3 + 2 * 1}{2} \right\rfloor + 1 = 3$$

Receptive Field

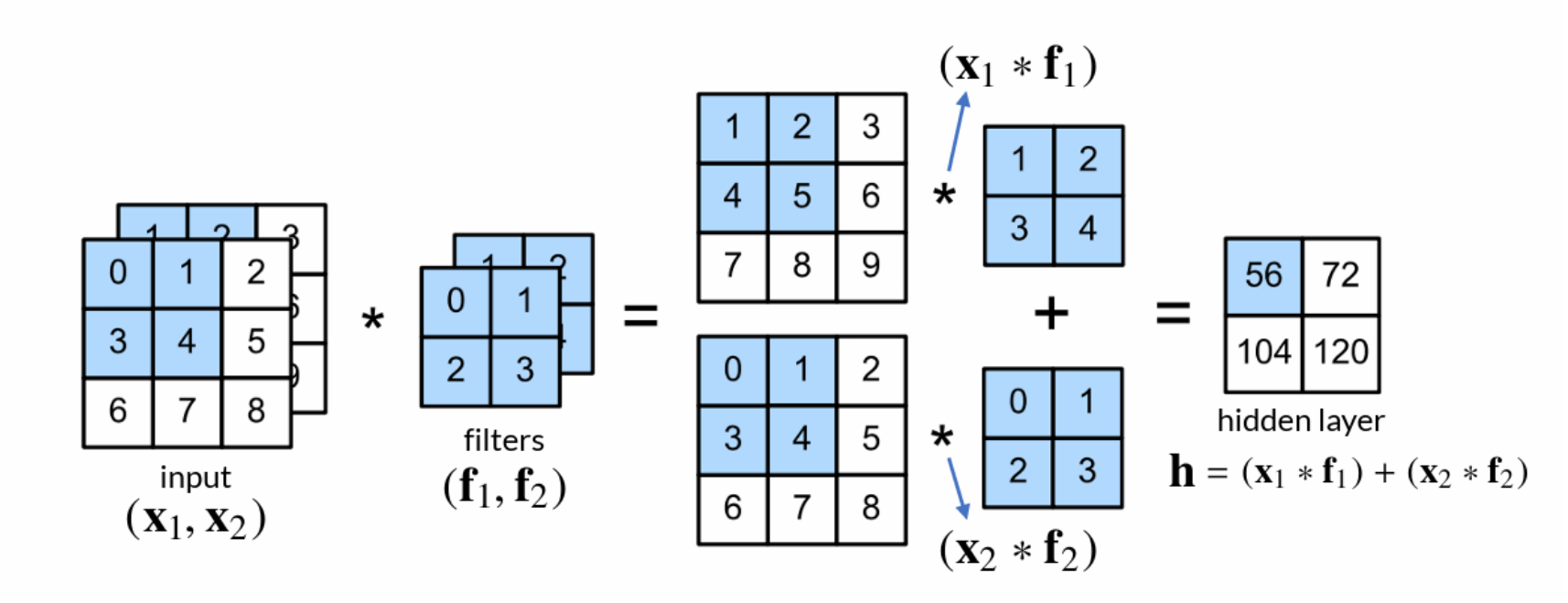
- 첫 3x3 Convolution의 결과 에서의 한 픽셀(파란색)은, 그 이전 레이어에서의 3x3 image의 정보만을 담고있다.
- 두번째 3x3 conv의 결과 (Layer 3의 노란색)은, 첫 레이어에서의 5x5 image의 정보만을 담고 있다.



Convolution Net = Convolution + Channel

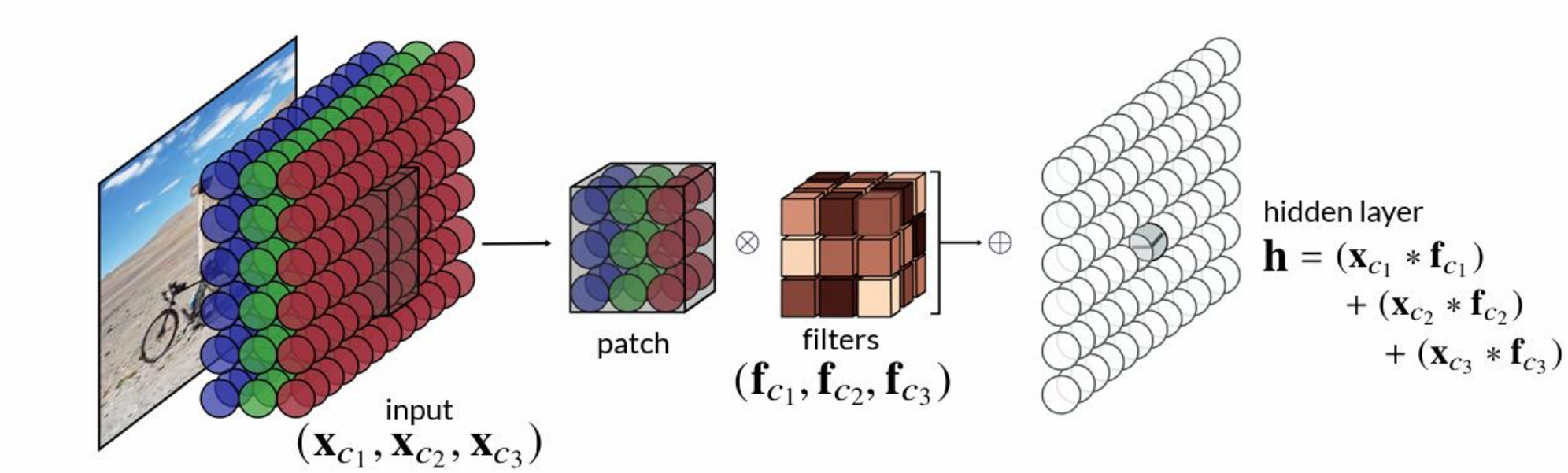


Convolution Net = Convolution + Channel



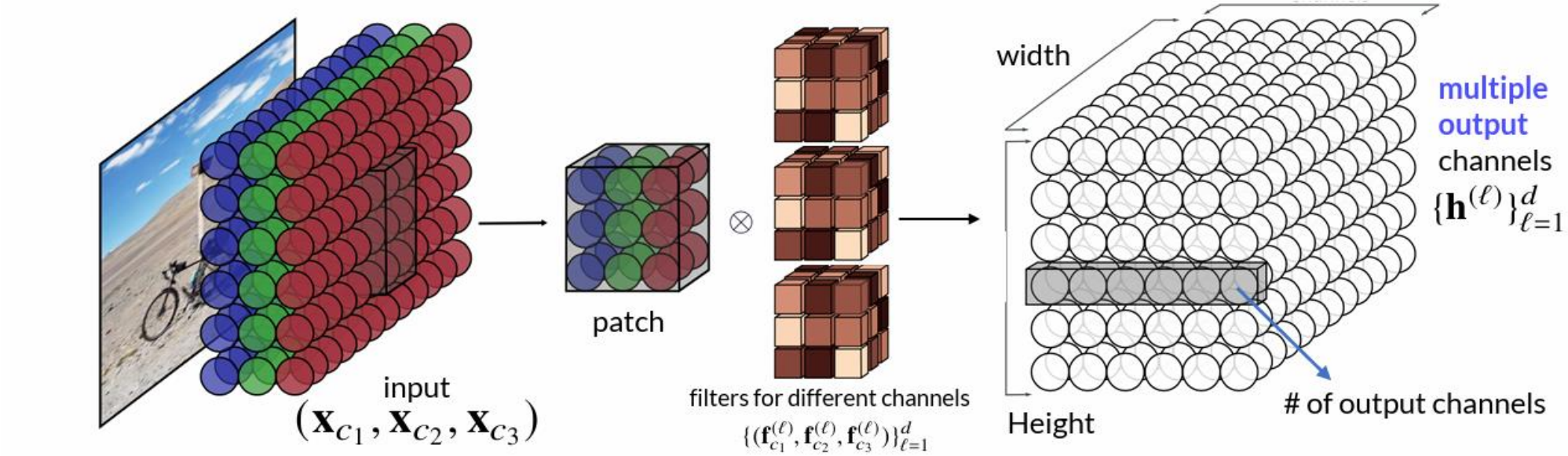
Convolution Net = Convolution + Channel

- RGB 이미지는 3개의 input channel이 있었다: Red, Green, and Blue
  - 하나의 Kernel(filter)는 각 위치의 세 channe을 patch 단위로 하나의 값으로 압축하여 spatial하게 위치시킨다.



Convolution Net = Convolution + Channel

- RGB 이미지는 3개의 input channel이 있었다: Red, Green, and Blue
  - Convolutional nets은 이 채널들을 multiple kernel을 사용하여 output channels로 mapping한다.

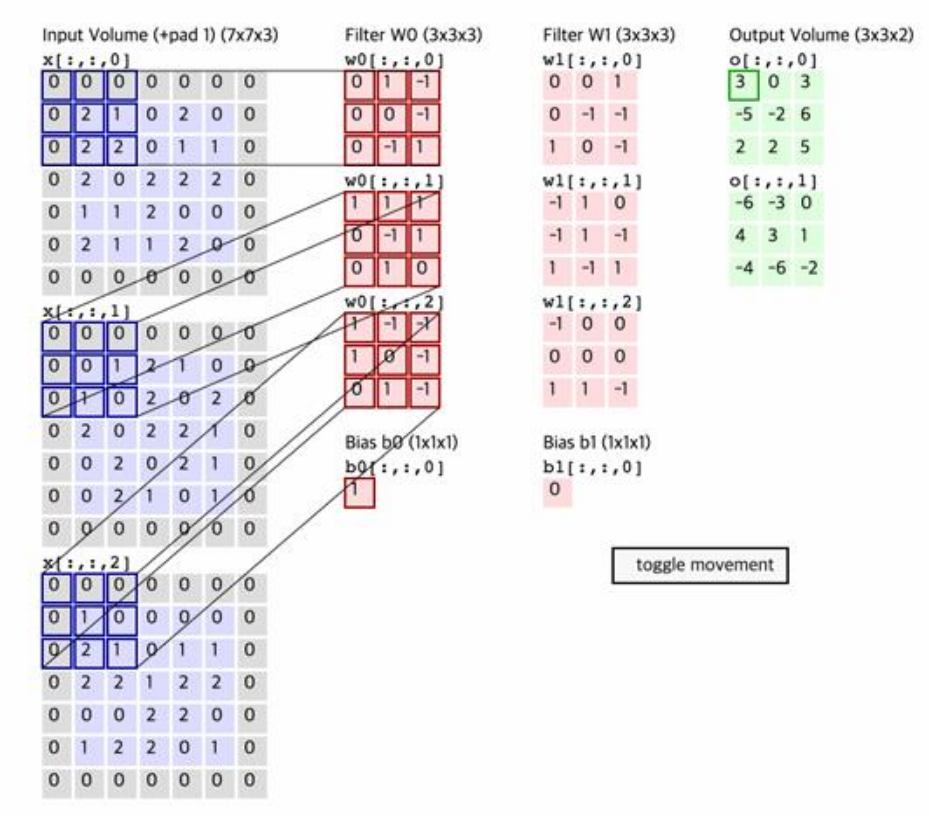


Number of Parameters of Convolutional Nets

- c_{in}: # of input channels
- c_{out}: # of output channels
- k_h, k_w: height and width oif filter(kernel)
- Filter shape: $c_{out} \times c_{in} \times k_h \times k_w$

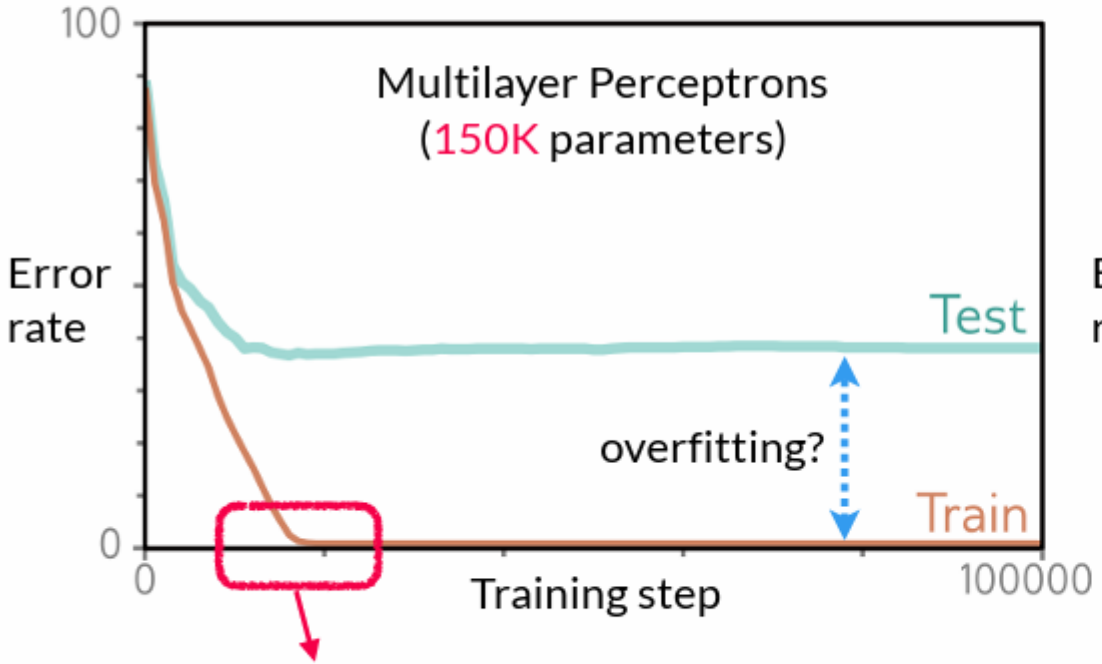
(Pop-quiz)

만약 $c_{out} \times c_{in} \times k_h \times k_w$의 필터와 (p_h, p_w)의 padding, (s_h, s_w)의 strid를 쓴다면 output의 차원은 어떻게 될까?

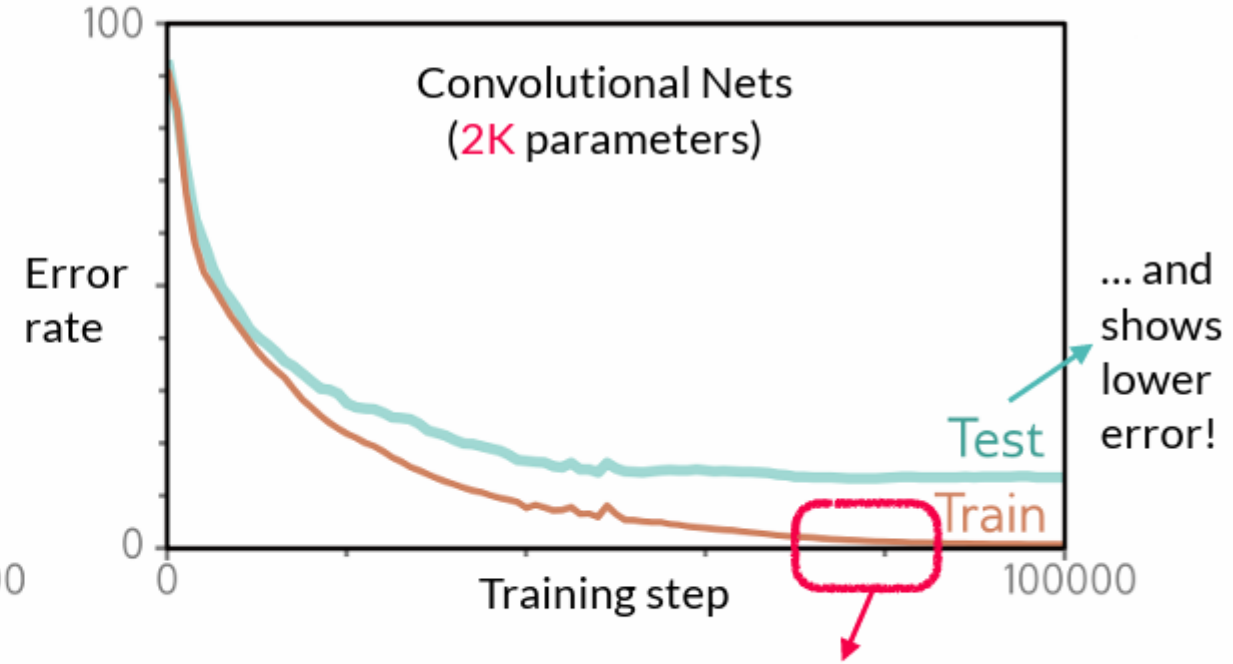


Example: MNIST

MLP도 MNIST dataset을 꽤 잘 predict하지만, CNN은 더 적은 파라미터로도 효율적으로 학습할 수 있다!



MLPs can reduce the train error...



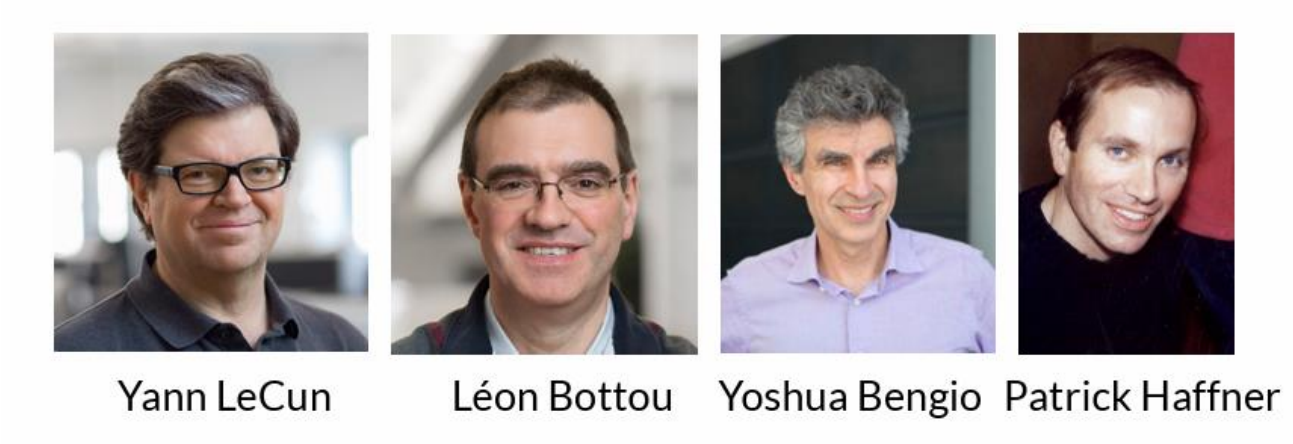
... but convolutional nets also have *expressive* power!

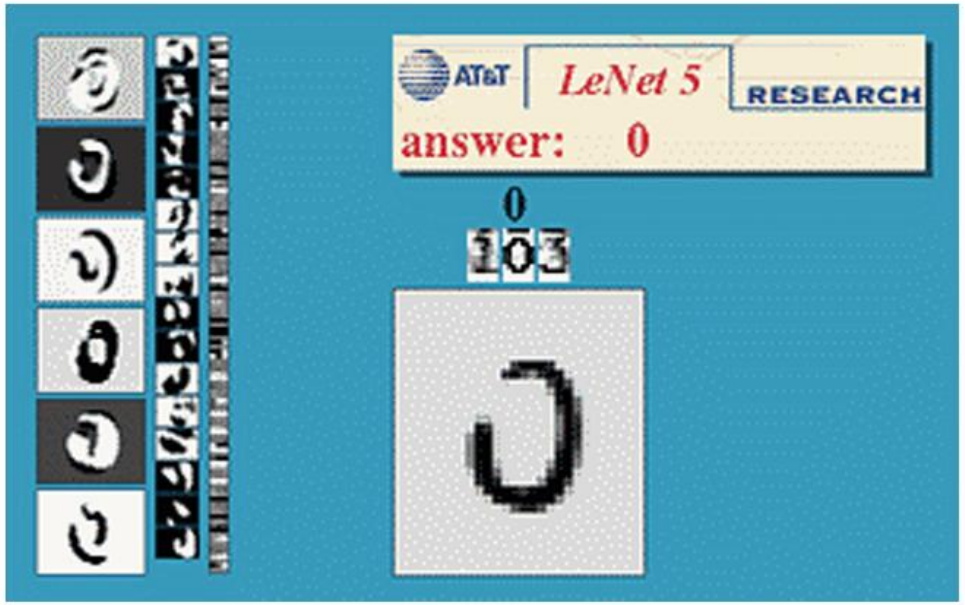
CNN이 MLP보다
generalization 능력이 더 뛰어나다!

..visual domain에서

LeNet

- CNN이 사용된 최초의 모델이다.
  - MLP에 비해 파라미터 수를 줄였다.
  - Image를 그리드로 처리하였다.
  - CNN을 역전파(backprop)로 학습하였다.
- ATM 기계에서 숫자를 인식하는데 사용되었다.





Gradient-Based Learning Applied to Document Recognition, Yann Lecun et al., Proc. of the IEEE (1998)

Missing Ingredients

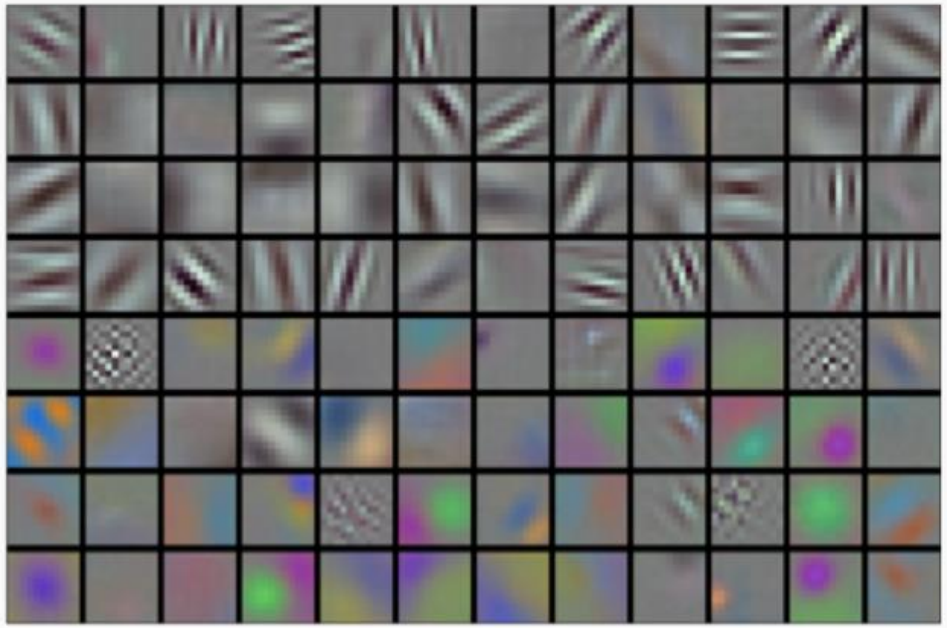
- 데이터의 부족
  - 딥러닝 모델은 많은 양을 데이터를 필요로 하지만 1998년에는 그러한 환경이 아니였다.
  - ImageNet은 2009년 Fei-Fei Li에 의해 release된다.
  - ImageNet competition은 컴퓨터비전과 머신러닝의 비약적 발전에 기여한다.

- 하드웨어의 부족
  - 수 백 epochs의 계산과 무거운 linear algebra 연산을 위해서는 좋은 GPU가 필요하다.

AlexNet

- 2012년까지 representation은 hand-crafted 알고리즘에 의해 계산되었다.
  - SIFT, SURF, HOG, …
- 일부 연구자들은 학습 방식으로 data를 표현(represent)할 수 있을 것이라 믿었다.
- AlexNet은 전통적인 필터(커널) 방식을 따르는 feature extractor를 배운 셈이다.

Alex Krizhevsky Ilya Sutskever Geoffrey E. Hinton



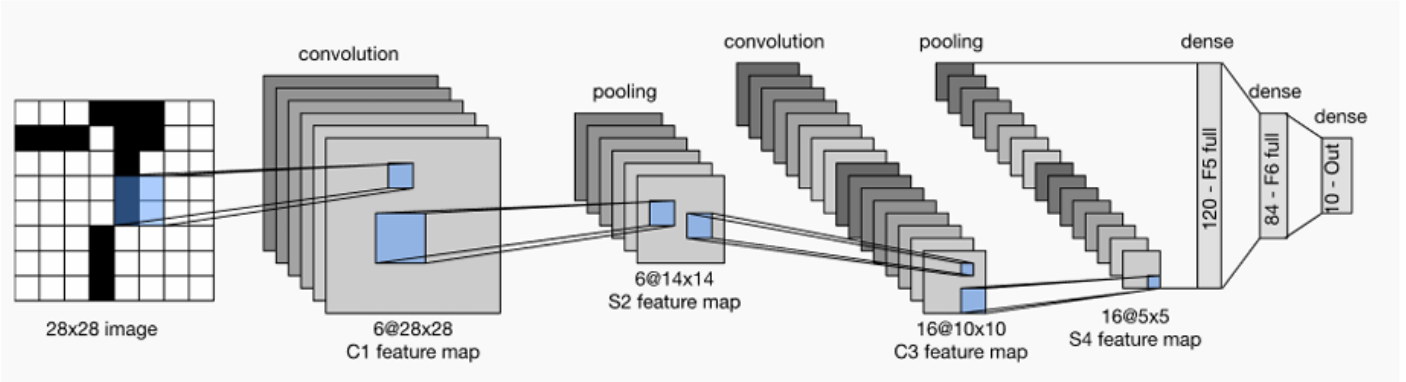
filters learned by AlexNet

ImageNet Classification with Deep Convolutional Neural Networks, Alex Krizhevsky et al., NIPS (2012)

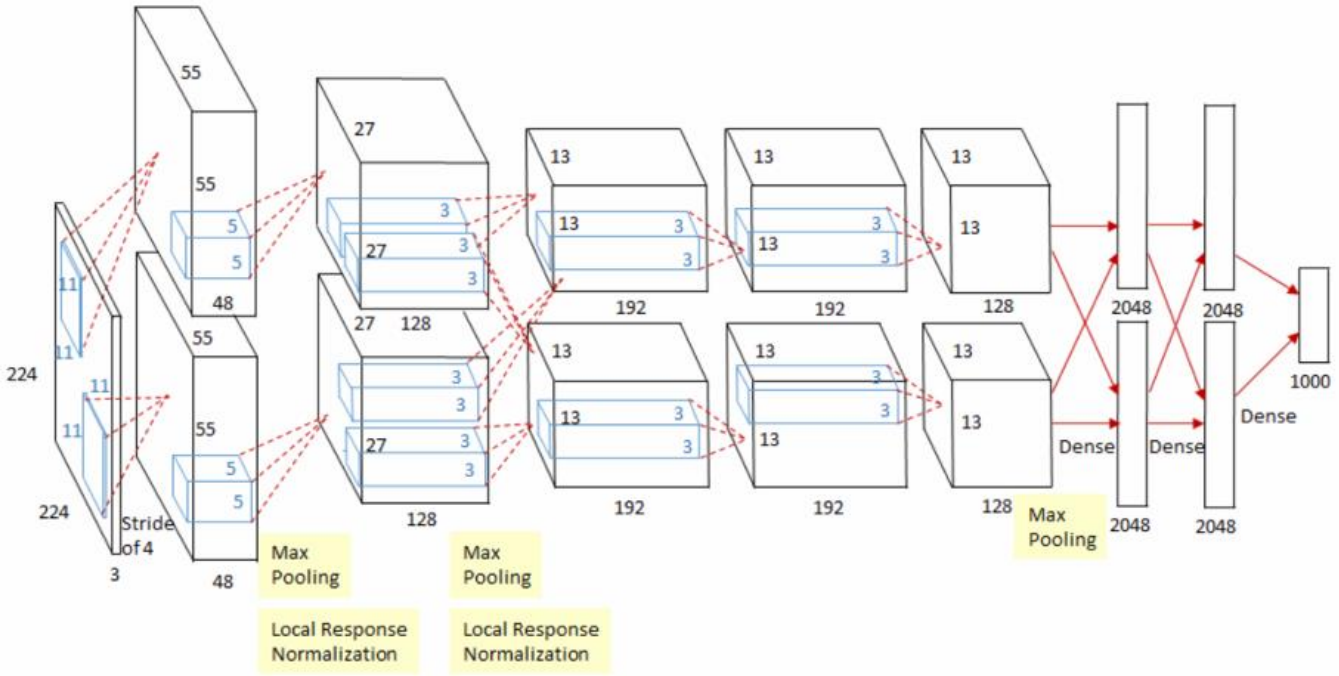
AlexNet

- LeNet
  - sigmoid
  - average pooling
- AlexNet (+2 GPUs)
  - ReLU
  - max pooling
  - dropout
  - augmentation

LeNet

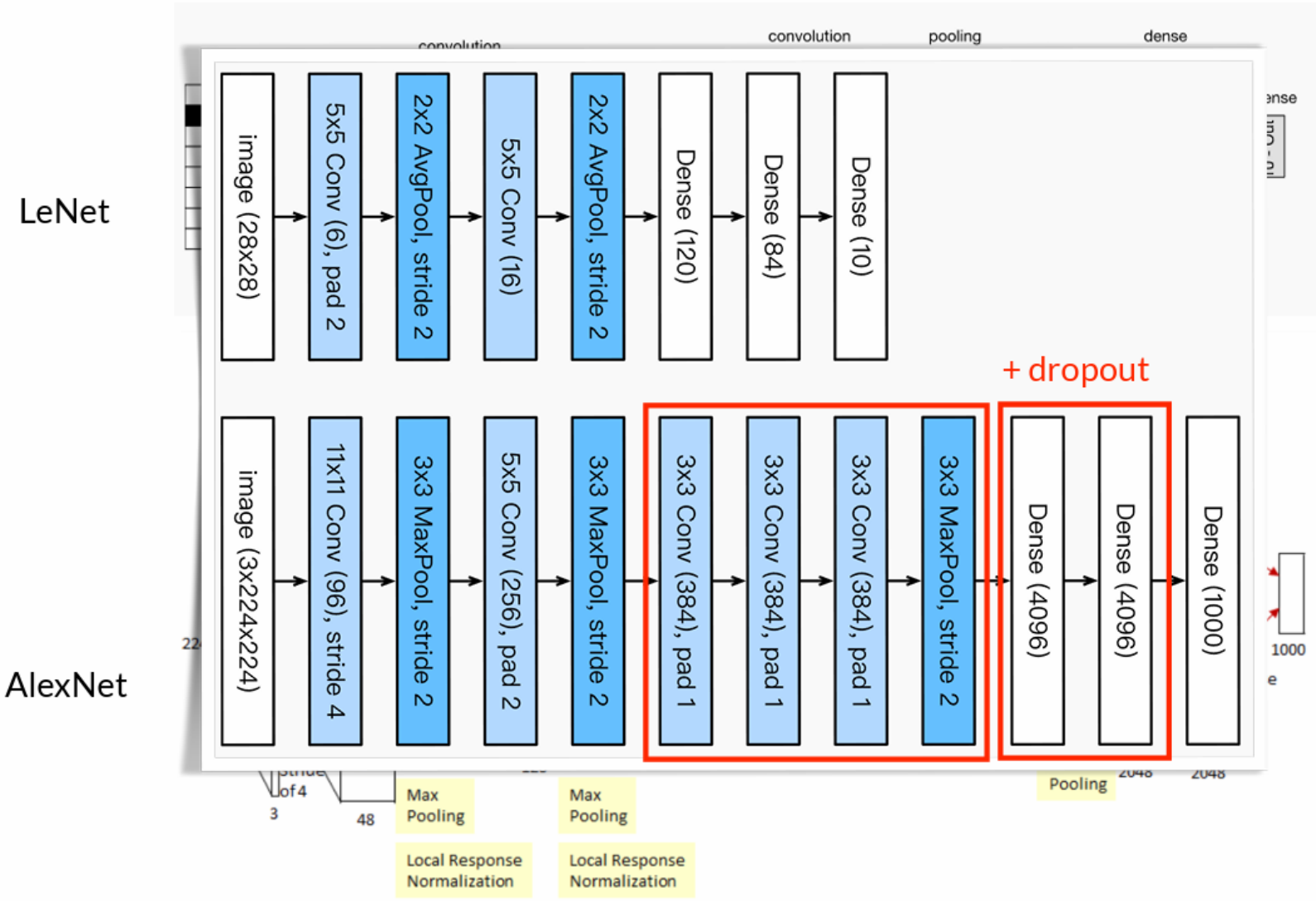


AlexNet



AlexNet

- LeNet
  - sigmoid
  - average pooling
- AlexNet (+2 GPUs)
  - ReLU
  - max pooling
  - dropout
  - augmentation

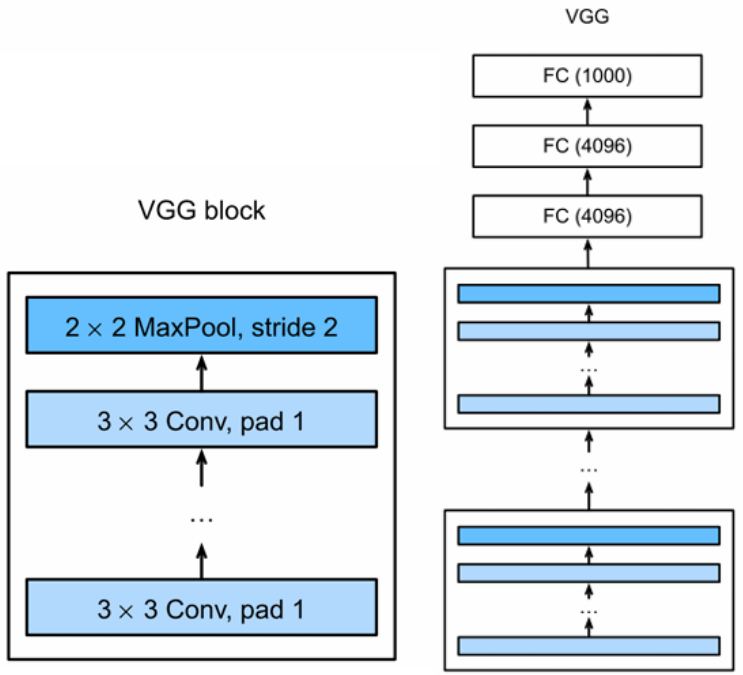


VGG

- Visual Geometry Group of Oxford Univ.
- '블록(block)을 단위로 이를 반복해 층을 쌓아보자'는 아이디어를 제시했다.
- 기본 블록은 CNN으로 구성된다.
  - Conv2D() + ReLU() + Pooling()
  - vgg_block은 convolutional layers의 sequence로 구성되어 있다.

    [Conv2D(out, in, padding=1), ReLU()] x N + MaxPool2D(3, stride=2)

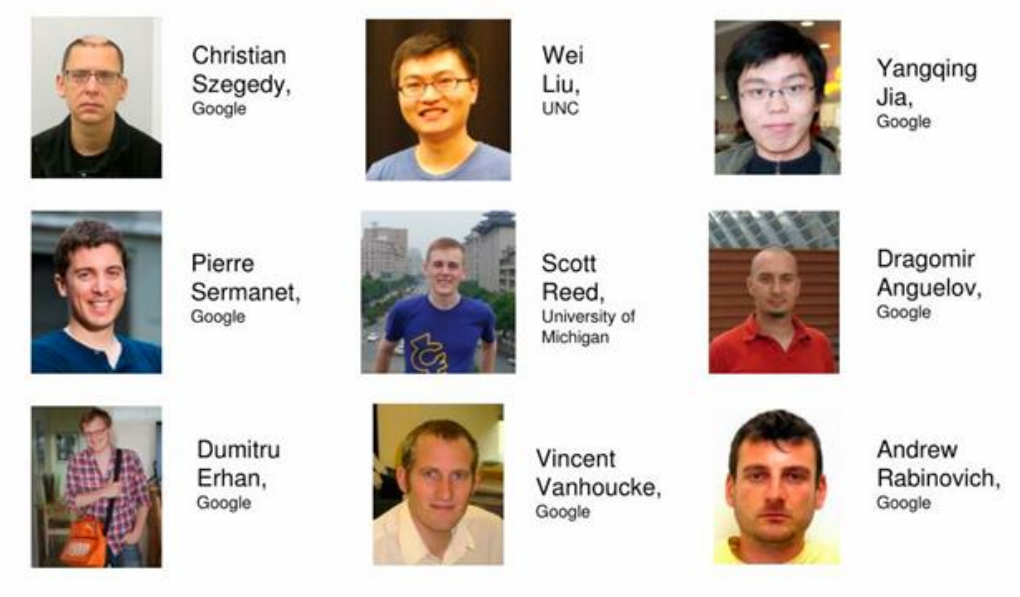
Karen Simonyan Andrew Zisserman

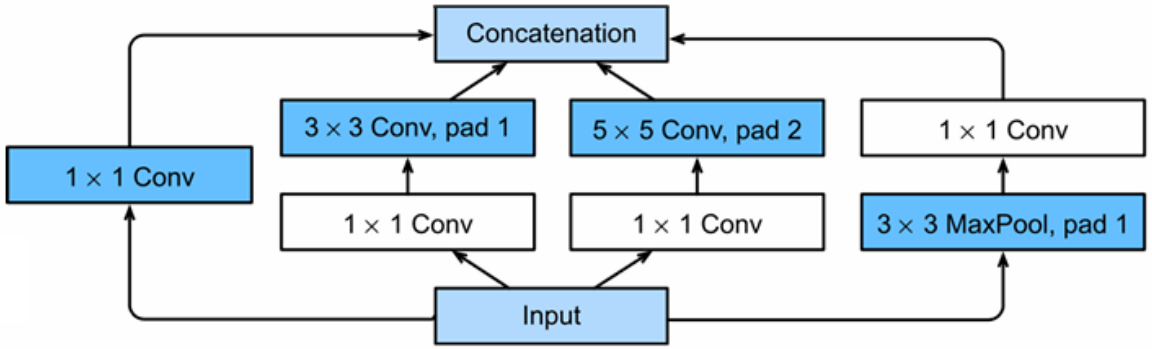


Very Deep Convolutional Networks for Large-Scale Image Recognition, Simonyan & Zisserman, ICLR (2015)

Inception

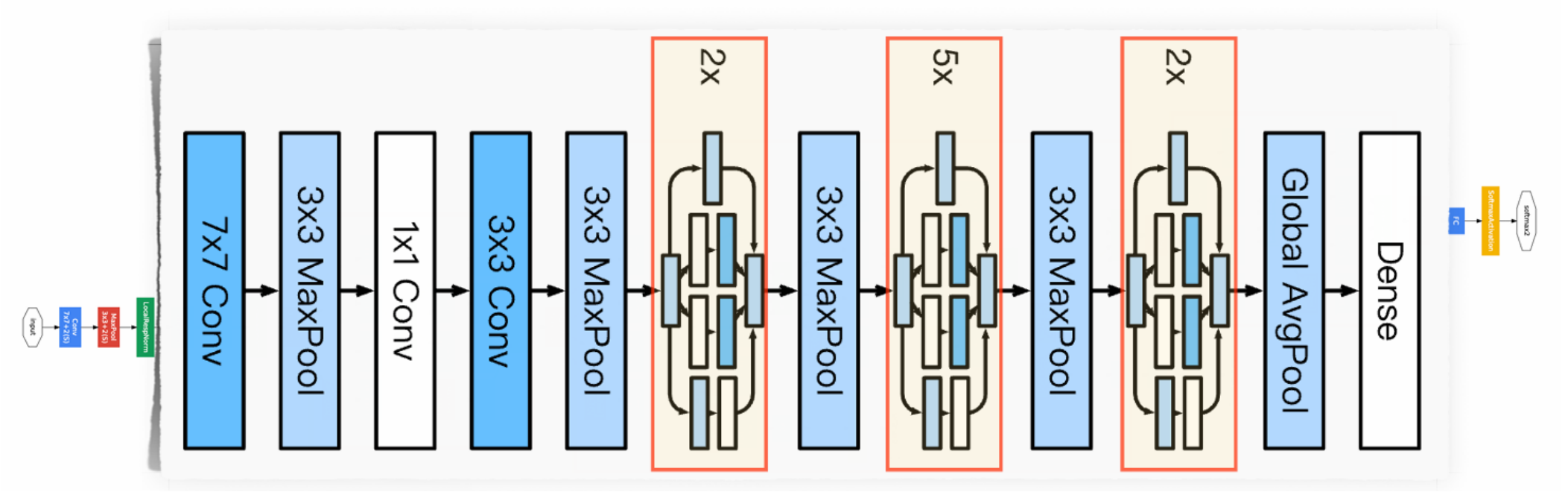
- 구글은 GoogLeNet으로 2014년 ImageNet challenge를 우승했다.
  - NiN(Network in Network)을 결합했다.
  - 다양한 필터의 조합으로 다른 모델에 비해 우위에 섰다.
- GoogLeNet은 Inception blocks를 이용했다.
  - 네 개의 parallel paths로 구성된다.
  - 여러 필터 사이즈의 convolution으로 구성된다.





Going deeper with convolutions, Szegedy et al., CVPR (2015)

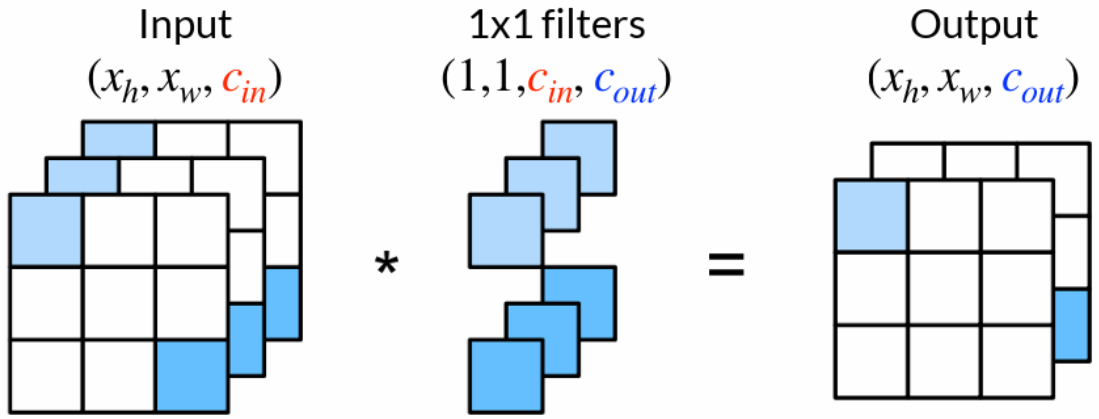
Inception Architecture



22개 layer로 구성된 Inception architecture는 **딥**러닝 시대로의 진입을 상징적으로 보여주었다

1x1 Convolution Layer

- 1x1 Conv layer는 보통 channel 차원을 맞춰주기 위해 사용한다.
  - 채널 차원을 맞출 때 사용되는 가장 대표적인 방법이다.
  - 출력의 각 요소는 입력의 동일 위치에 대한 linear combination이다.
  - 즉, channel mixing이다.
  - 이는 곧 Fully Connected Layer와 같다.



Pop Quiz

- 어떤 CNN 모델이 가장 파라미터가 적을까?
    1. AlexNet (8-layers)
    2. VGGNet (19-layers)
    3. GoogLeNet (22-layers)

Pop Quiz

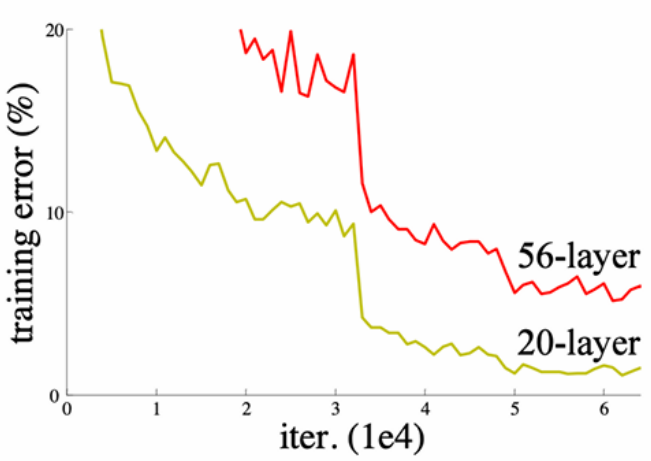
- 어떤 CNN 모델이 가장 파라미터가 적을까?
  1. AlexNet (8-layers) → 60M
  2. VGGNet (19-layers) → 138M
  3. GoogLeNet (22-layers) → 4M

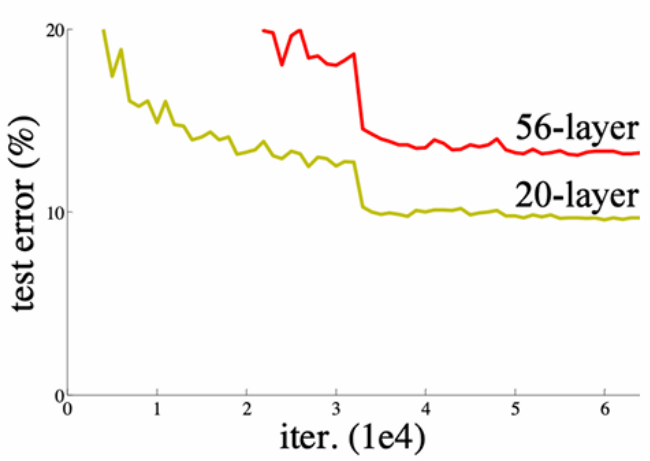
더 많은 layer가 있다하여 파라미터가 더 많은 것은 아니다!

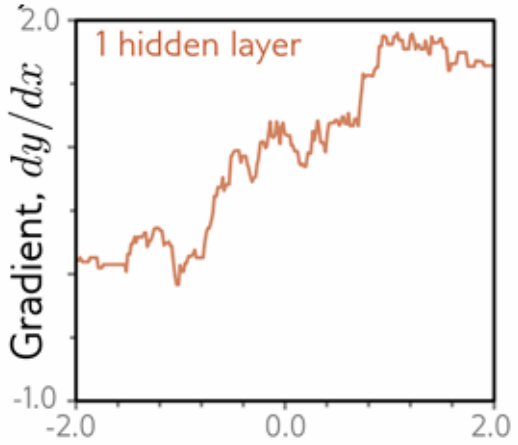
더 많은 layers를 쌓을 수 있을까?

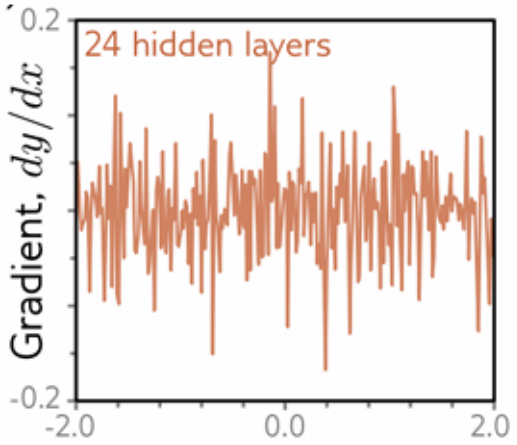
Deeper layers suffer optimization problem!

- Neural networks가 깊어질수록 학습하기 힘들어진다.
  - (주의) Overfitting 때문에 발생하는 것이 아니다.
  - 아래 figure를 보면 56-layer일 때가 20-layer일 때보다 test error가 더 높다.
  - Deep networks일수록 gradient가 굉장히 빠르게 변하기 때문이다.





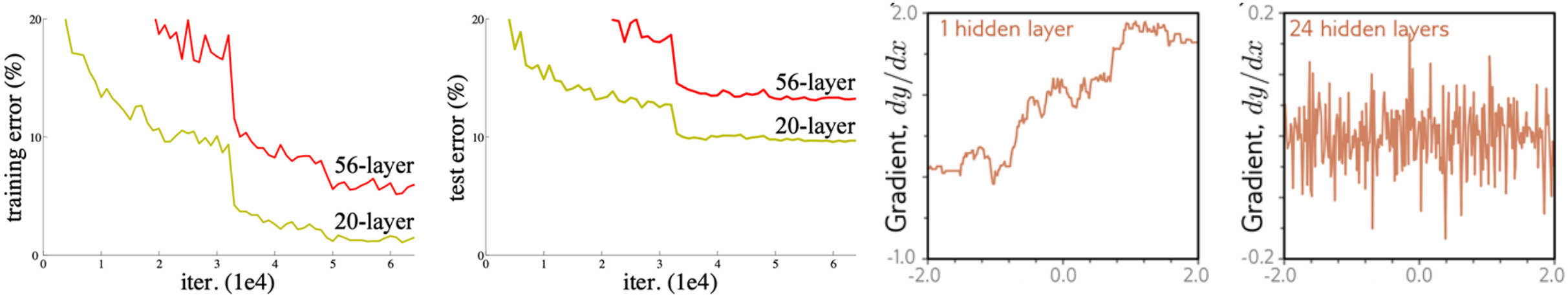




Deep Residual Learning for Image Recognition, He et al., CVPR (2016)

Deeper layers suffer optimization problem!

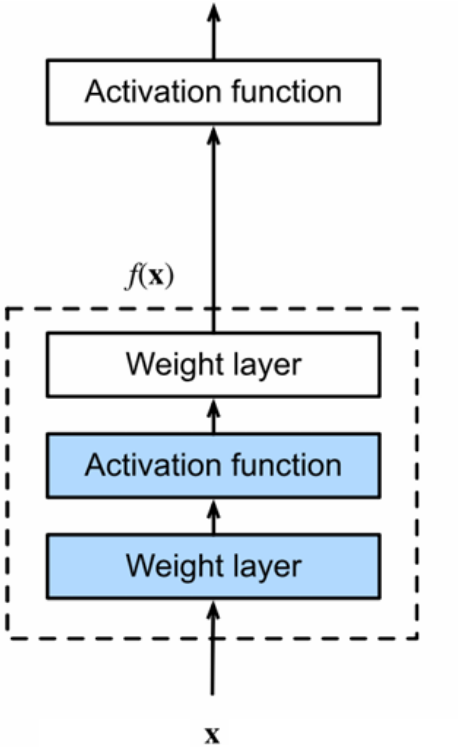
- Neural networks가 깊어질수록 학습하기 힘들어진다.
  - (주의) Overfitting 때문에 발생하는 것이 아니다.
  - 아래 figure를 보면 56-layer일 때가 20-layer일 때보다 test error가 더 높다.
  - 학습이 힘든 이유는 네트워크가 깊을수록 gradient가 굉장히 빠르게 변하기 때문이다.



Deep Residual Learning for Image Recognition, He et al., CVPR (2016)

Deeper layers suffer optimization problem

- Nonlinear function 다음에 identity map을 더하자 : $f(\mathbf{x})$

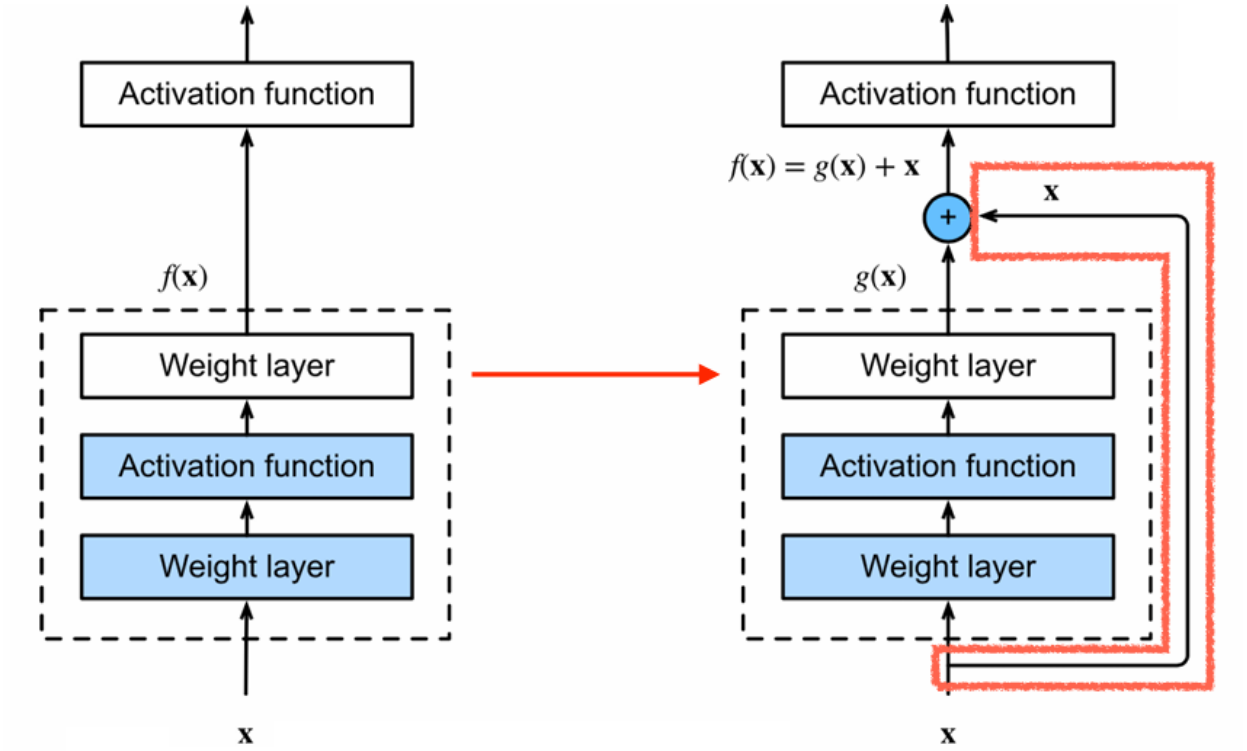


Plain Neural Nets

$$\mathbf{h}_1 = \mathrm{Net}_1(\mathbf{x}; \theta_1)$$
$$\mathbf{h}_2 = \mathrm{Net}_2(\mathbf{h}_1; \theta_2)$$
$$\mathbf{h}_3 = \mathrm{Net}_3(\mathbf{h}_2; \theta_3)$$
$$\vdots$$

Solution: Add Skip Connection

- Nonlinear function 다음에 identity map을 더하자 : $\mathrm{f}(\mathrm{x}) \rightarrow \mathrm{x} + f(\mathrm{x})$



Network w/ *Skip Connection*

$$\mathbf{h}_1 = \mathbf{x} + \mathrm{Net}_1(\mathbf{x}; \theta_1)$$
$$\mathbf{h}_2 = \mathbf{h}_1 + \mathrm{Net}_2(\mathbf{h}_1; \theta_2)$$
$$\mathbf{h}_3 = \mathbf{h}_2 + \mathrm{Net}_3(\mathbf{h}_2; \theta_3)$$
$$\vdots$$

이것을

skip connection

또는

residual connection

이라고 부른다.

How residual connection works?

Plain Neural Nets

$$\mathbf{h}_1 = \mathrm{Net}_1(\mathbf{x}; \theta_1)$$
$$\mathbf{h}_2 = \mathrm{Net}_2(\mathbf{h}_1; \theta_2)$$
$$\mathbf{h}_3 = \mathrm{Net}_3(\mathbf{h}_2; \theta_3)$$
$$\vdots$$

$$\mathbf{y} = \mathrm{Net}_L(\mathrm{Net}_{L-1}(\cdots(\mathrm{Net}_2(\mathrm{Net}_1(\mathbf{x}; \theta_1); \theta_2)\cdots); \theta_{L-1}); \theta_L)$$

$$\longrightarrow \frac{\partial \mathbf{y}}{\partial \mathbf{h}_1} = \frac{\partial \mathbf{y}}{\partial \mathbf{h}_L}\frac{\partial \mathbf{h}_L}{\partial \mathbf{h}_{L-1}}\cdots\frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1}$$

앞쪽 layer에서의 파라미터 변화가 뒤쪽 layer의 gradient에 큰 영향을 미친다

How residual connection works?

Plain Neural Nets

$$\mathbf{h}_1 = \text{Net}_1(\mathbf{x}; \theta_1)$$
$$\mathbf{h}_2 = \text{Net}_2(\mathbf{h}_1; \theta_2)$$
$$\mathbf{h}_3 = \text{Net}_3(\mathbf{h}_2; \theta_3)$$
$$\vdots$$

$$\mathbf{y} = \text{Net}_L(\text{Net}_{L-1}(\cdots(\text{Net}_2(\text{Net}_1(\mathbf{x}; \theta_1); \theta_2)\cdots); \theta_{L-1}); \theta_L)$$

$$\longrightarrow \frac{\partial \mathbf{y}}{\partial \mathbf{h}_1} = \frac{\partial \mathbf{y}}{\partial \mathbf{h}_L}\frac{\partial \mathbf{h}_L}{\partial \mathbf{h}_{L-1}}\cdots\frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1}$$

Network w/ *Skip Connection*

$$\mathbf{h}_1 = \mathbf{x} + \text{Net}_1(\mathbf{x}; \theta_1)$$
$$\mathbf{h}_2 = \mathbf{h}_1 + \text{Net}_2(\mathbf{h}_1; \theta_2)$$
$$\mathbf{h}_3 = \mathbf{h}_2 + \text{Net}_3(\mathbf{h}_2; \theta_3)$$
$$\vdots$$

Residual connection 덕분에 deep layers에서도 shattered gradient를 덜 겪을 수 있다

$$\mathbf{y} = \mathbf{x} + \text{Net}_1(\mathbf{x}; \theta_1) + \text{Net}_2(\mathbf{x} + \text{Net}_1(\mathbf{x}; \theta_1); \theta_2) + \cdots$$

$$\longrightarrow \frac{\partial \mathbf{y}}{\partial \mathbf{h}_1} = \mathbf{I} + \frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1} + \left(\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_1} + \frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1}\right) + \cdots$$

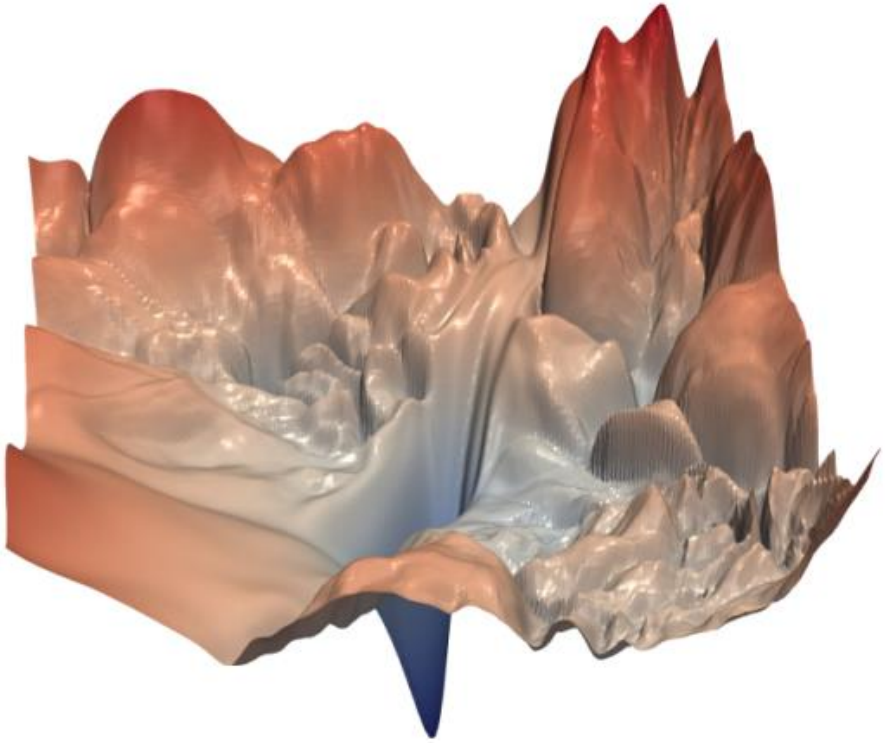
How residual connection works?

Plain Ne

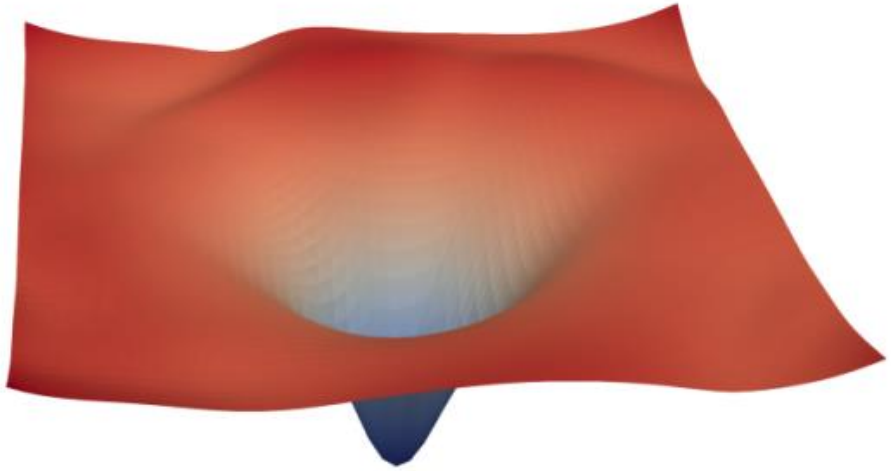
$\mathbf{h}_1 = \text{Net}_1$
$\mathbf{h}_2 = \text{Net}_2$
$\mathbf{h}_3 = \text{Net}_3$

Network w/ S

$\mathbf{h}_1 = \mathbf{x} + \text{N}$
$\mathbf{h}_2 = \mathbf{h}_1 +$
$\mathbf{h}_3 = \mathbf{h}_2 +$



(a) without skip connections



(b) with skip connections

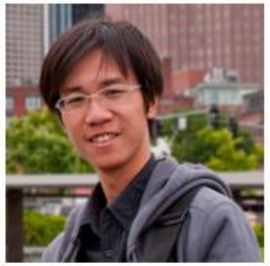
$); \theta_L)$

덜 겪을 수 있다

..

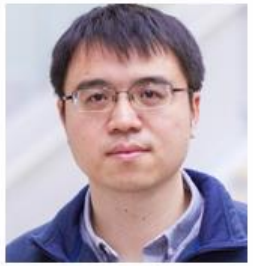
$+ \cdots$

ResNet

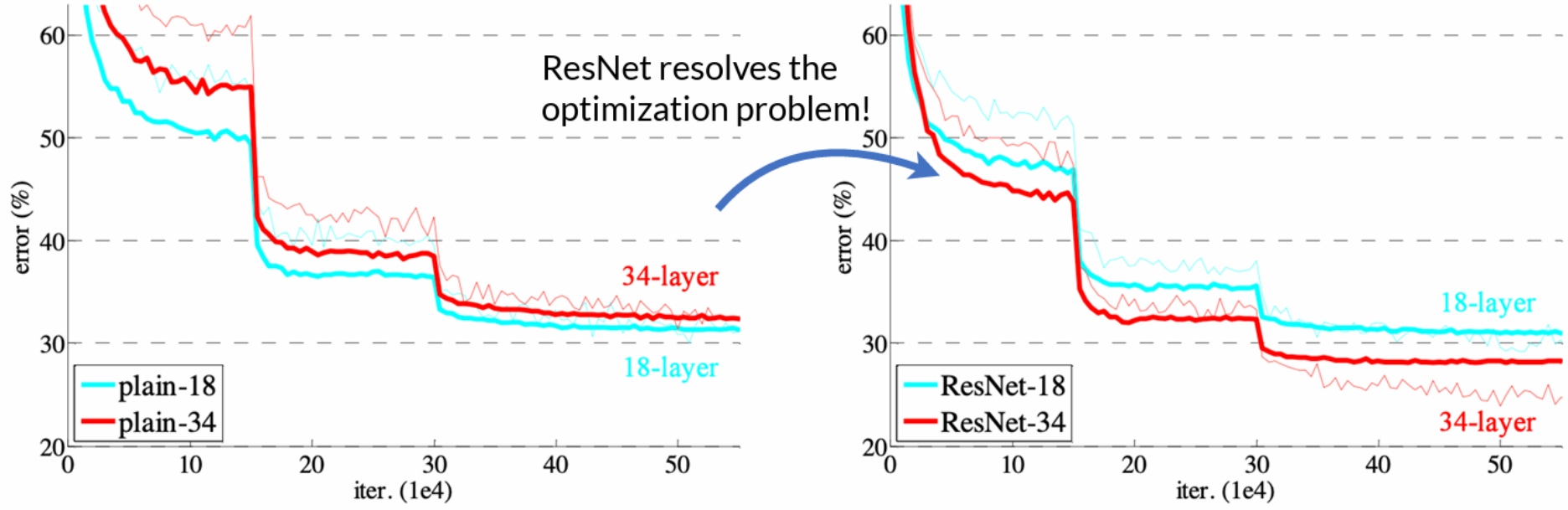

Kaiming He


Xiangyu Zhang


Shaoqing Ren


Jian Sun

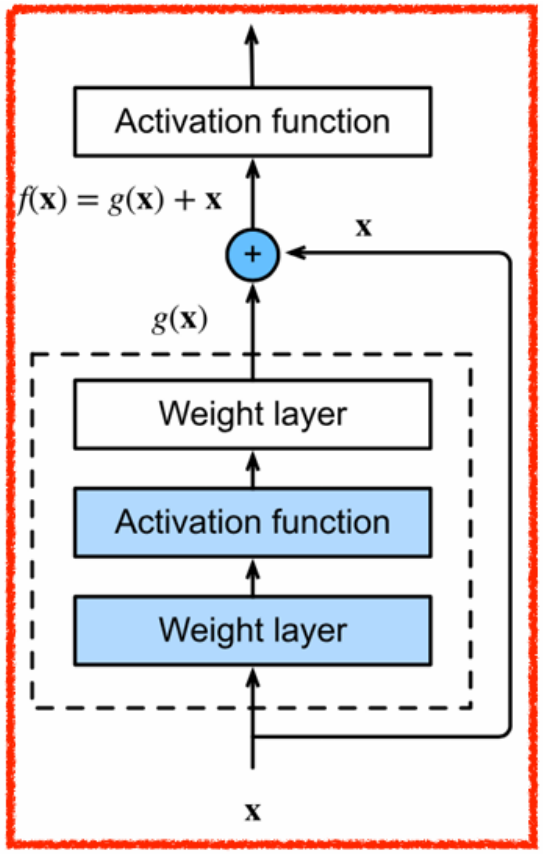
- 2016년 Microsoft Research Asia의 Kaiming He 박사 팀은 residual connection을 이용한 ResNet을 발표한다.
- 이 모델은 2015년 ILSVRC 대회를 우승했으며, 이공계 전체에서 역사상 가장 많이 인용된 논문이자, CNN 아키텍쳐 연구의 종결을 가져왔다.
- 현재에도 모든 딥러닝 연구 및 개발의 baseline으로써 활발히 사용되고 있다.

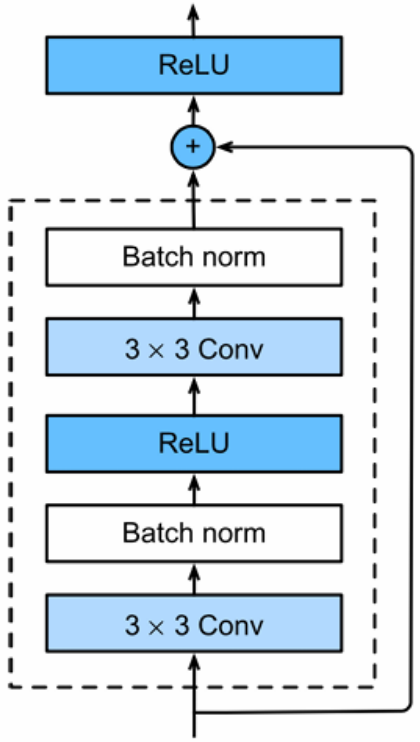


Deep Residual Learning for Image Recognition, He et al., CVPR (2016)

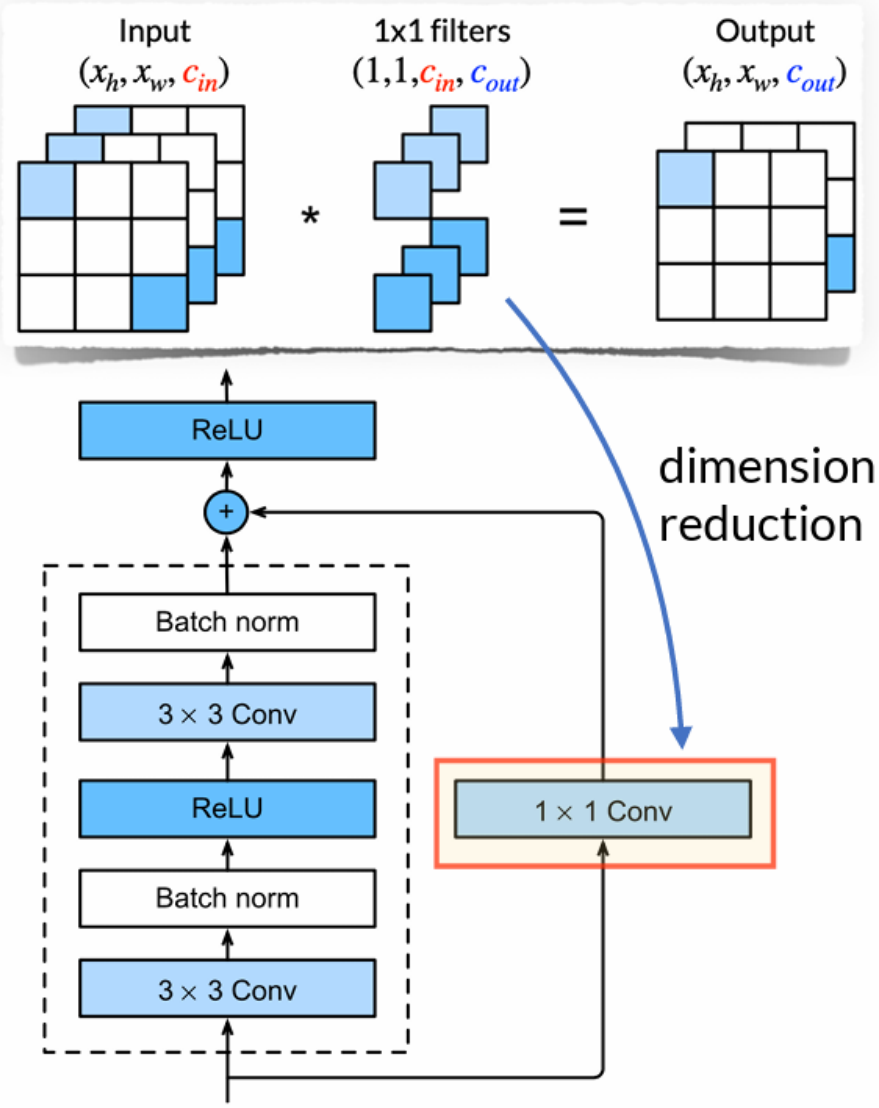
ResNet: Residaul Blocks

- Residual block: Conv + ReLU + Conv + Skip





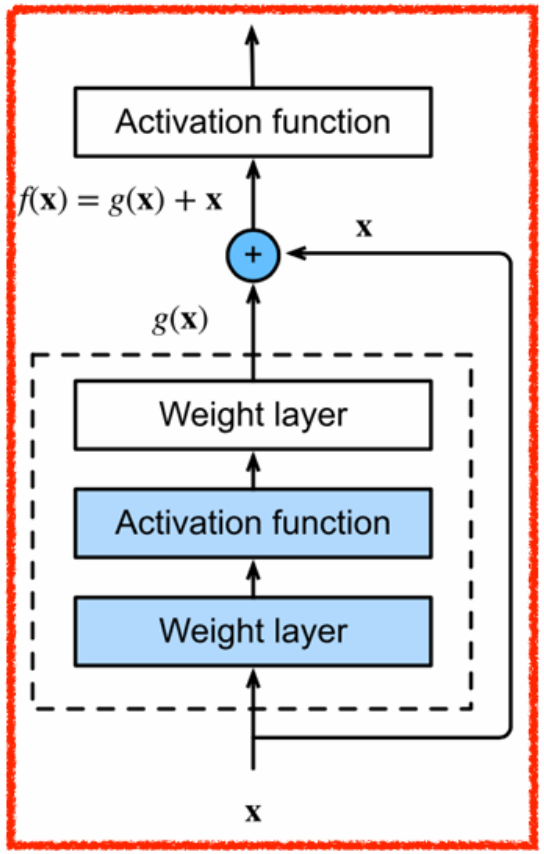
type 1: direct shortcut

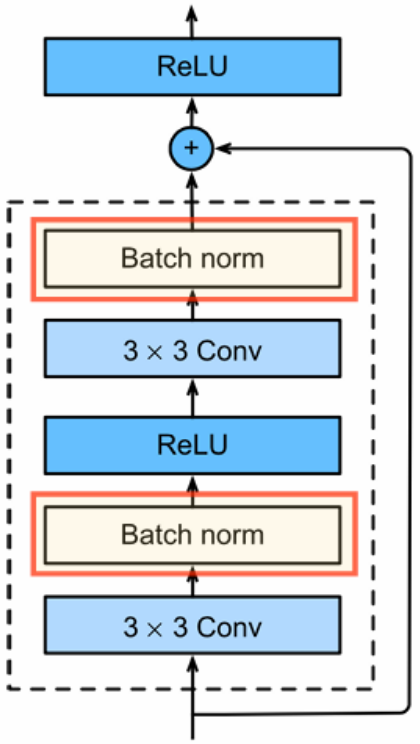


type 2: projection shortcut

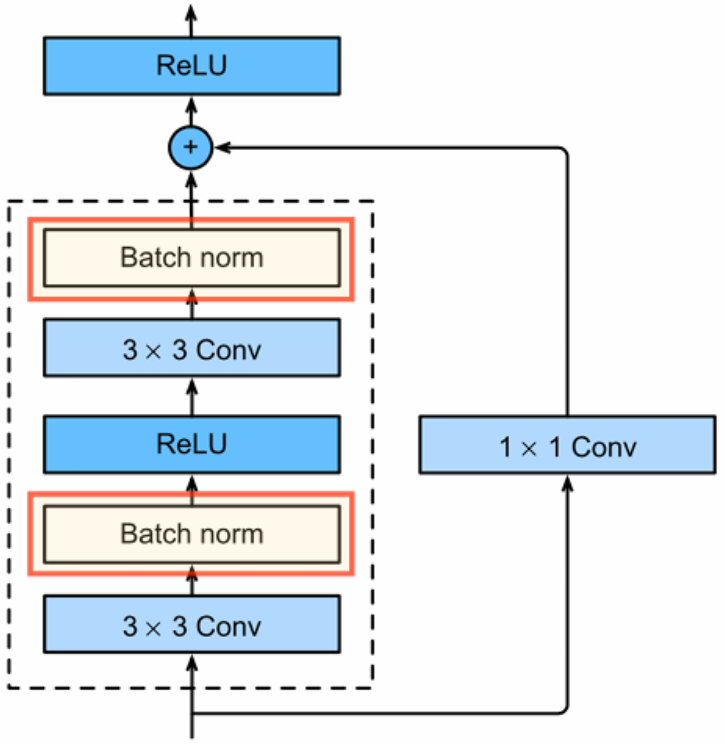
ResNet: Residaul Blocks + BatchNorm

- Residual block: Conv + BatchNorm + ReLU + Conv + BatchNorm + Skip





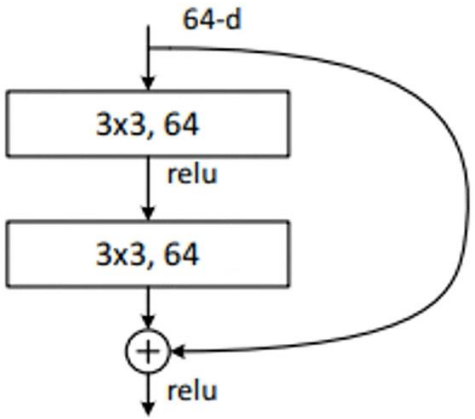
type 1: direct shortcut



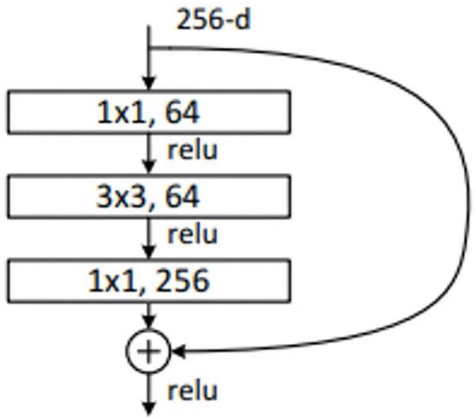
type 2: projection shortcut

ResNet: BottleNeck Block

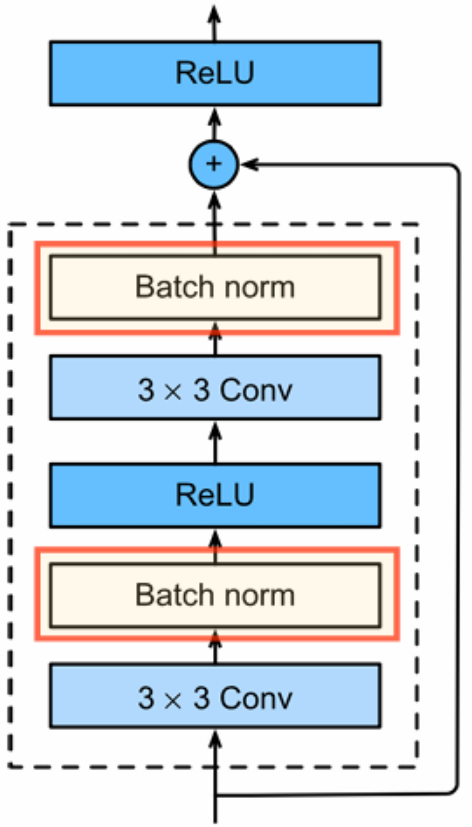
- 두 개의 1x1 Conv Block 이용, Channel을 축소시켰다가 다시 증가
- 연산량 상에서의 이점



Standard



BottleNeck



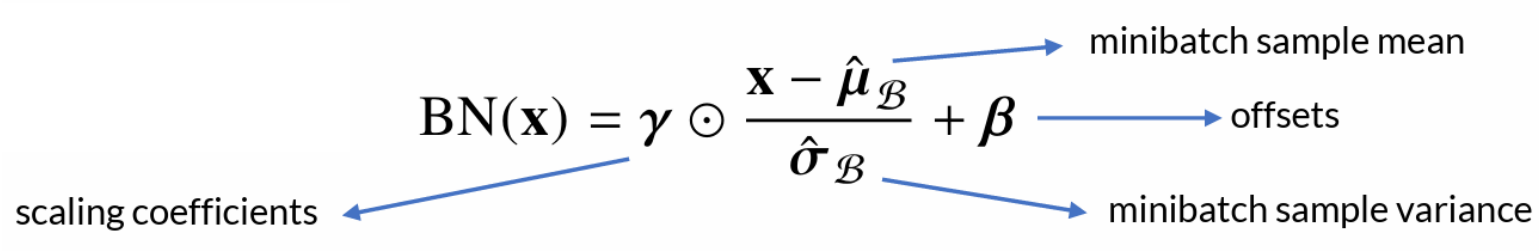
ResNets

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of block

Batch Normalization

- 2016년 Loffe & Szegedy는 BatchNorm을 제안하였다.
- BatchNorm은 deep nets의 수렴을 가속화시키는 효과적인 방법이다.
  - 최초의 의도는 regularization이 아니였다.
  - BatchNorm과 residual blocks를 함께 사용하여 100 layers가 넘는 매우 깊은 네트워크도 훈련할 수 있게 되었다.

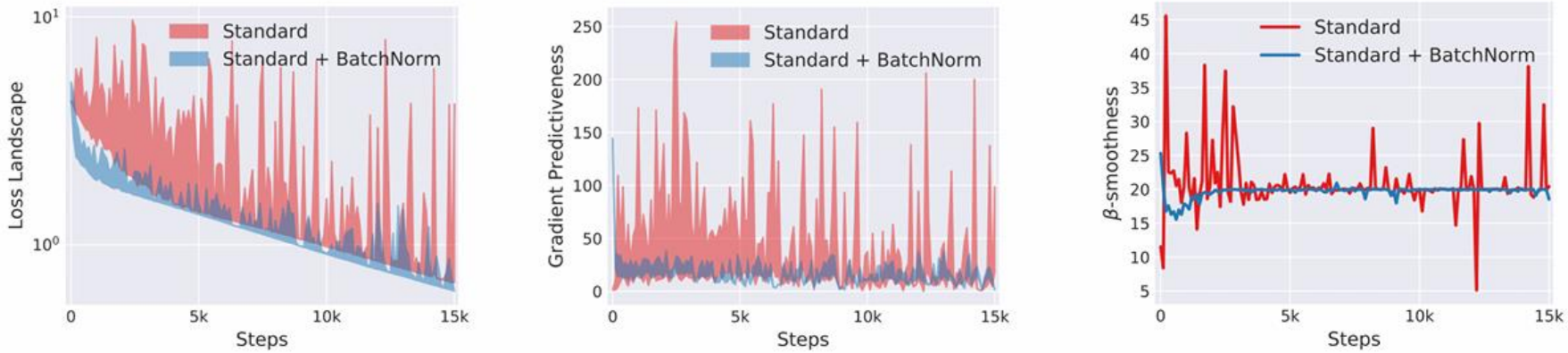


(Tips) batch size가 작을 경우 가급적 BatchNorm을 사용하지 마세요 (중심극한정리..)

Batch Normalization, Ioffe & Szegedy, ICML (2016)

How BatchNorm works?

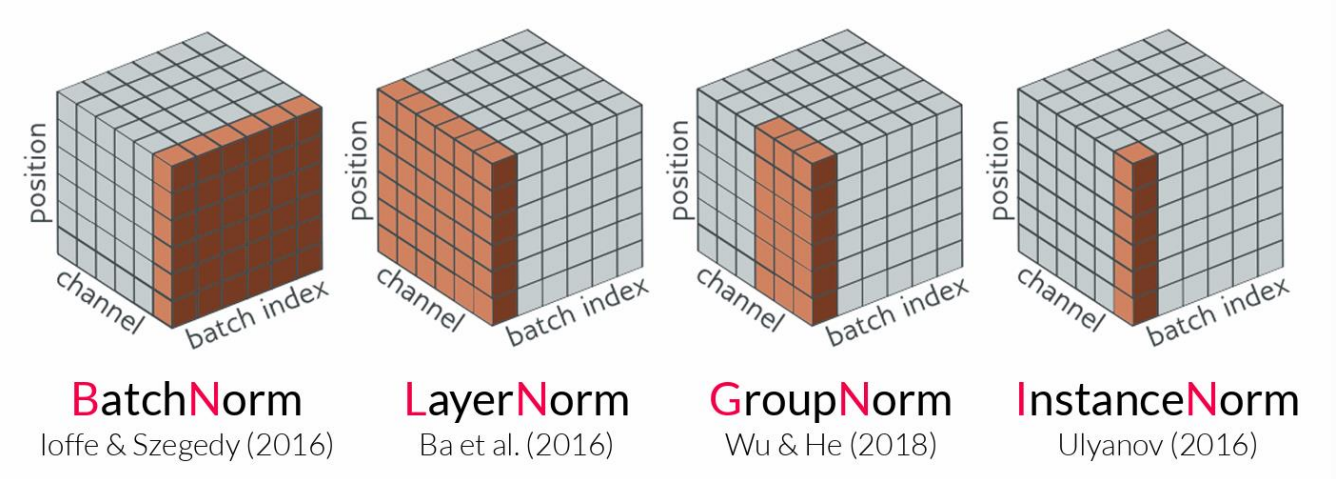
- 원 저자(Ioffe & Szegedy)는 내부 공변량(internal covariant shift)의 해소로 이를 설명하였다.
  - 그러나 BatchNorm은 내부 공변량을 오히려 더 증가시키는 것이 밝혀졌다.
- 최근 연구들은 BatchNorm이 loss surface를 smooth하게 만든다고 주장한다 → 아직 증명되지 않음!
- 아직 이론적으로 BatchNorm이 잘 동작하는 이유는 증명되지 않았다.



How Does Batch Normalization Help Optimization?, Santurkar et al., NeurIPS (2018)

Normalization Layers

- Activation-based Layers
  - Batch Normalization (BN)
  - Layer Normalization (LN)
  - Group Normalization (GN)
  - Instance Normalization (IN)
- Parametric Layers
  - Weight Normalization (WN)
  - Scaled Weight Standardization (SWS)



모든 activation-based normalizers은 forward propagation을 안정화시키는 특성을 갖고 있다.

이 중 특이하게 GN은 gradient explosion을 줄여주는 것이 알려져 있다.

Beyond BatchNorm: Towards a Unified Understanding of Normalization in Deep Learning, Lubana et al., NeurIPS (2021)

실습:
VGG-16과 ResNet을 구현해보자!

KUGODS